Intel® OpenSource HD Graphics Programmer's Reference Manual (PRM)

Volume 1 Part 1: Graphics Core (SandyBridge)

For the 2011 Intel Core Processor Family

May 2011

Revision 1.0

NOTICE:

This document contains information on products in the design phase of development, and Intel reserves the right to add or remove product features at any time, with or without changes to this open source documentation.

Doc Ref #: IHD-OS-V1 Pt1 – 05 11

Creative Commons License

You are free to Share — to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

No Derivative Works. You may not alter, transform, or build upon this work.

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

Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The Sandy Bridge chipset family, Havendale/Auburndale chipset family, Intel® 965 Express Chipset Family, Intel® G35 Express Chipset, and Intel® 965GMx Chipset Mobile Family Graphics Controller may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order. I2C is a two-wire communications bus/protocol developed by Philips. SMBus is a subset of the I2C bus/protocol and was developed by Intel.

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

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

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

Copyright © 2011, Intel Corporation. All rights reserved.

Contents

1. Introduction ... 6
1.1 Devices and Device Tag Definitions ... 7
1.2 Reserved Bits and Software Compatibility ... 7
1.3 Terminology ... 8
2. Graphics Device Overview ... 16
2.1 Graphics Memory Controller Hub (GMCH) ... 16
2.2 Graphics Processing Unit (GPU) ... 17
3. Graphics Processing Engine (GPE) ... 19
3.1 Introduction ... 19
3.2 Overview ... 19
3.2.1 Command Stream (CS) Unit ... 21
3.2.2 GPE Function IDs ... 22
3.3 Pipeline Selection ... 23
3.4 URB Allocation ... 24
3.5 Constant URB Entries (CURBEs) ... 24
3.6 Memory Object Control State ... 24
3.6.1 MEMORY_OBJECT_CONTROL_STATE ... 25
3.7 Memory Access Indirection ... 26
3.7.1 STATE_BASE_ADDRESS ... 27
3.8 Instruction and State Prefetch ... 33
3.8.1 STATE_PREFETCH ... 34
3.9 System Thread Configuration ... 35
3.9.1 STATE_SIP ... 35
3.10 Command Ordering Rules ... 35
3.10.1 PIPELINE_SELECT ... 36
3.10.2 PIPE_CONTROL ... 36
3.10.3 URB-Related State-Setting Commands ... 37
3.10.4 Common Pipeline State-Setting Commands ... 37
3.10.5 3D Pipeline-Specific State-Setting Commands ... 37
3.10.6 Media Pipeline-Specific State-Setting Commands ... 38
3.10.7 URB_FENCE (URB Fencing & Entry Allocation) ... 38
3.10.8 CONSTANT_BUFFER (CURBE Load) ... 39
3.10.9 3DPRIMITIVE ... 39
3.10.10 MEDIA_OBJECT ... 39
4. Video Codec Engine ... 40
4.1 Video Command Streamer (VCS) ... 41
5. Graphics Command Formats ... 43
5.1 Command Formats ... 43
5.1.1 Memory Interface Commands ... 44
5.1.2 2D Commands ... 44
5.1.3 3D/Media Commands ... 44
5.1.4 Video Codec Commands ... 44
5.2 Command Map ... 47
5.2.1 Memory Interface Command Map ... 47
5.2.2 Video Codec Command Map ... 52
6. Register Address Maps ... 56
6.1 Graphics Register Address Map ... 56
6.1.1 Memory and I/O Space Registers [DevSNB+] ... 56

6.1.2 Graphics Register Memory Address Map [DevSNB] 57
6.2 VGA and Extended VGA Register Map 72
6.2.1 VGA and Extended VGA I/O and Memory Register Map 72
6.3 Indirect VGA and Extended VGA Register Indices 74
7. Memory Data Formats 77
7.1 Memory Object Overview 77
7.1.1 Memory Object Types 77
7.2 Channel Formats 78
7.2.1 Unsigned Normalized (UNORM) 78
7.2.2 Gamma Conversion (SRGB) 78
7.2.3 Signed Normalized (SNORM) 79
7.2.4 Unsigned Integer (UINT/USCALED) 79
7.2.5 Signed Integer (SINT/SSCALED) 79
7.2.6 Floating Point (FLOAT) 79
7.3 Non-Video Surface Formats 83
7.3.1 Surface Format Naming 83
7.3.2 Intensity Formats 83
7.3.3 Luminance Formats 83
7.3.4 R1_UNORM (same as R1_UINT) and MONO8 84
7.3.5 Palette Formats 85
7.4 Compressed Surface Formats 87
7.4.1 FXT Texture Formats 87
7.4.2 BC4 100
7.4.3 BC5 101
7.5 Video Pixel/Texel Formats 103
7.5.1 Packed Memory Organization 103
7.5.2 Planar Memory Organization 105
7.6 Surface Memory Organizations 106
7.7 Graphics Translation Tables 106
7.8 Hardware Status Page 107
7.9 Instruction Ring Buffers 107
7.10 Instruction Batch Buffers 107
7.11 Display, Overlay, Cursor Surfaces 107
7.12 2D Render Surfaces 108
7.13 2D Monochrome Source 108
7.14 2D Color Pattern 108
7.15 3D Color Buffer (Destination) Surfaces 108
7.16 3D Depth Buffer Surfaces 109
7.17 3D Separate Stencil Buffer Surfaces [DevILK+] 109
7.18 Surface Layout 110
7.18.1 Buffers 110
7.18.2 1D Surfaces 111
7.18.3 2D Surfaces 111
7.18.4 Cube Surfaces 115
7.18.5 3D Surfaces 116
7.19 Surface Padding Requirements 118
7.19.1 Sampling Engine Surfaces 118
7.19.2 Render Target and Media Surfaces 118
7.19.3 Register/State Context [DevSNB+] 119
7.19.4 The Per-Process Hardware Status Page 120
7.19.5 Register/State Context 120
7.19.6 The Per-Process Hardware Status Page 121
7.19.7 Overall Context Layout 122

7.19.8 Register/State Context 122
7.19.9 Pipelined State Page 129
7.19.10 Ring Buffer 129
7.19.11 The Per-Process Hardware Status Page 129

1. Introduction

The Intel® HD Graphics Open Source Programmer's Reference Manual (PRM) describes the architectural behavior and programming environment of the GEN chipset family. The Graphics Controller (GC) contains an extensive set of registers and instructions for configuration, 2D, 3D, and Video systems. The PRM describes the register, instruction, and memory interfaces and the device behaviors as controlled and observed through those interfaces. The PRM also describes the registers and instructions, and provides detailed bit/field descriptions.

The PRM is organized into four volumes:

PRM Volume 1: Graphics Core

This volume covers the overall Graphics Processing Unit (GPU), and not much detail on 3D, Media, or the core subsystem. Topics include the command streamer, context switching, and memory access (including tiling). Memory Data Formats can also be found here.

The volume also contains a chapter on the Graphics Processing Engine (GPE). The GPE is a collective term for 3D, Media, the subsystem, and the parts of the memory interface that are used by these units. Display, blitter, and their memory interfaces are *not* included in the GPE.

PRM Volume 2: 3D / Media

This volume includes description of the 3D and Media pipelines in detail. It contains details for all of the "fixed functions," including commands processed by the pipelines, fixed-function state structures, and a definition of the inputs (payloads) and outputs of the threads spawned by these units.

This volume also covers the single Media Fixed Function, VLD. It describes how to initiate generic threads using the thread spawner (TS). Generic threads will be used on the GEN family for the majority of media functions. Programmable kernels handle the algorithms for media functions such as IDCT, Motion Compensation, and Motion Estimation (used for encoding MPEG streams).

PRM Volume 3: Display Registers

Volume 3 describes the control registers for the display. These registers control the display, the overlay, and VGA.

PRM Volume 4: Subsystem and Cores/ Shared Functions

The GEN subsystem contains the programmable cores, or Executable Units (EUs), and the "shared functions" that are shared by more than one EU and perform I/O functions and complex math functions.

The shared functions consist of the sampler:

- Extended math unit
- Data port (the interface to memory for 3D and media)
- Unified Return Buffer (URB)

- The Message Gateway used by EU threads to signal each other

The EUs use messages to send data to and receive data from the subsystem; the messages are described with the shared functions. The generic message 'send EU instruction' is described with the rest of the instructions in the Instruction Set Architecture (ISA) chapters.

The latter part of this volume describes the GMHC core, or EU, and the associated instructions used to program it. The instruction descriptions make up an Instruction Set Architecture, or ISA. The ISA describes all of the instructions that the GEN core can execute, along with the registers that are used to store local data.

1.1 Devices and Device Tag Definitions

Device "Tags," used in various parts of this document as aliases for the device names, are listed in the following table. Note that stepping information is sometimes appended to the device tag, e.g., [**DevCTG-A**]. Information without any device tagging is applicable to all devices.

The table below lists the standard for defining all device and stepping tags:

Table 1-1 Supported Chipsets

Device Abbreviation	Product Name	Program Name	SKU
DevSNB	TBD	SandyBridge, aka DevSNB	GT1
	TBD		GT2.0
	TBD		GT2.1
	TBD		GT2

NOTES:

1. Unless otherwise specified, the information in this document applies to all of the devices mentioned in Table 1-1. For information that does not apply to all devices, the Device Tag is used.
2. Throughout the PRM, references to "All" in a project field refters to all devices in Table 1-1.
3. Stepping information is sometimes appended to the device tag (e.g., [DevCTG-A]). Information without any device tagging is applicable to all devices/steppings.
5. A shorthand is used to identify all devices/steppings prior to the device/stepping that the item pertains. Notations and Conventions.

1.2 Reserved Bits and Software Compatibility

In many register, instruction, and memory layout descriptions, certain bits are marked as "Reserved". When bits are marked as reserved, it is essential for compatibility with future devices that the software treat these bits as having a future, though unknown, effect. The behavior of reserved bits should be

regarded as undefined *and unpredictable*. Software should follow these guidelines in dealing with reserved bits:

1. Do not depend on the states of any reserved bits when testing values of registers that contain such bits.

2. Mask out the reserved bits before testing.

3. Do not depend on the states of any reserved bits when storing to an instruction or to a register.

4. When loading a register or formatting an instruction, always load the reserved bits with the values indicated in the documentation (if any), or reload them with the values previously read from the register.

1.3 Terminology

Term	Abbr.	Definition
3D Pipeline	--	One of the two pipelines supported in the GPE. The 3D pipeline is a set of fixed-function units arranged in a pipelined fashion, which process 3D-related commands by spawning EU threads. Typically this processing includes rendering primitives. See *3D Pipeline*.
Adjacency	--	One can consider a single line object as existing in a strip of connected lines. The neighboring line objects are called "adjacent objects", with the non-shared endpoints called the "adjacent vertices." The same concept can be applied to a single triangle object, considering it as existing in a mesh of connected triangles. Each triangle shares edges with three other adjacent triangles, each defined by an non-shared adjacent vertex. Knowledge of these adjacent objects/vertices is required by some object processing algorithms (e.g., silhouette edge detection). See *3D Pipeline*.
Application IP	AIP	Application Instruction Pointer. This is part of the control registers for exception handling for a thread. Upon an exception, hardware moves the current IP into this register and then jumps to SIP.
Architectural Register File	ARF	A collection of architecturally visible registers for a thread such as address registers, accumulator, flags, notification registers, IP, null, etc. ARF should not be mistaken as just the address registers.
Array of Cores	--	Refers to a group of GEN EUs, which are physically organized in two or more rows. The fact that the EUs are arranged in an array is (to a great extent) transparent to CPU software or EU kernels.
Binding Table	--	Memory-resident list of pointers to surface state blocks (also in memory).
Binding Table Pointer	BTP	Pointer to a binding table, specified as an offset from the Surface State Base Address register.
Bypass Mode	--	Mode where a given fixed function unit is disabled and forwards data down the pipeline unchanged. Not supported by all FF units.
Byte	B	A numerical data type of 8 bits, B represents a signed byte integer.

Term	Abbr.	Definition
Child Thread		A branch-node or a leaf-node thread that is created by another thread. It is a kind of thread associated with the media fixed function pipeline. A child thread is originated from a thread (the parent) executing on an EU and forwarded to the Thread Dispatcher by the TS unit. A child thread may or may not have child threads depending on whether it is a branch-node or a leaf-node thread. All pre-allocated resources such as URB and scratch memory for a child thread are managed by its parent thread.
Clip Space	--	A 4-dimensional coordinate system within which a clipping frustum is defined. Object positions are projected from Clip Space to NDC space via "perspecitive divide" by the W coordinate, and then viewport mapped into Screen Space
Clipper	--	3D fixed function unit that removes invisible portions of the drawing sequence by discarding (culling) primitives or by "replacing" primitives with one or more primitives that replicate only the visible portion of the original primitive.
Color Calculator	CC	Part of the Data Port shared function, the color calculator performs fixed-function pixel operations (e.g., blending) prior to writing a result pixel into the render cache.
Command	--	Directive fetched from a ring buffer in memory by the Command Streamer and routed down a pipeline. Should not be confused with instructions which are fetched by the instruction cache subsystem and executed on an EU.
Command Streamer	CS or CSI	Functional unit of the Graphics Processing Engine that fetches commands, parses them and routes them to the appropriate pipeline.
Constant URB Entry	CURBE	A UE that contains "constant" data for use by various stages of the pipeline.
Control Register	CR	The read-write registers are used for thread mode control and exception handling for a thread.
Data Port	DP	Shared function unit that performs a majority of the memory access types on behalf of GEN programs. The Data Port contains the render cache and the constant cache and performs all memory accesses requested by GEN programs except those performed by the Sampler. See DataPort.
Degenerate Object	--	Object that is invisible due to coincident vertices or because does not intersect any sample points (usually due to being tiny or a very thin sliver).
Destination	--	Describes an output or write operand.
Destination Size		The number of data elements in the destination of a GEN SIMD instruction.
Destination Width		The size of each of (possibly) many elements of the destination of a GEN SIMD instruction.
Double Quad word (DQword)	DQ	A fundamental data type, DQ represents 16 bytes.
Double word (DWord)	D or DW	A fundamental data type, D or DW represents 4 bytes.
Drawing Rectangle	--	A screen-space rectangle within which 3D primitives are rendered. An objects screen-space positions are relative to the Drawing Rectangle origin.
End of Block	EOB	A 1-bit flag in the non-zero DCT coefficient data structure indicating the end of an 8x8 block in a DCT coefficient data buffer.
End Of Thread	EOT	a message sideband signal on the Output message bus signifying that the message requester thread is terminated. A thread must have at least one SEND instruction with the EOT bit in the message descriptor field set in order to properly terminate.

Term	Abbr.	Definition
Exception	--	Type of (normally rare) interruption to EU execution of a thread's instructions. An exception occurrence causes the EU thread to begin executing the System Routine which is designed to handle exceptions.
Execution Channel	--	
Execution Size	ExecSize	Execution Size indicates the number of data elements processed by a GEN SIMD instruction. It is one of the GEN instruction fields and can be changed per instruction.
Execution Unit	EU	Execution Unit. An EU is a multi-threaded processor within the GEN multi-processor system. Each EU is a fully-capable processor containing instruction fetch and decode, register files, source operand swizzle and SIMD ALU, etc. An EU is also referred to as a GEN Core.
Execution Unit Identifier	EUID	The 4-bit field within a thread state register (SR0) that identifies the row and column location of the EU a thread is located. A thread can be uniquely identified by the EUID and TID.
Execution Width	ExecWidth	The width of each of several data elements that may be processed by a single GEN SIMD instruction.
Extended Math Unit	EM	A Shared Function that performs more complex math operations on behalf of several EUs.
FF Unit	--	A Fixed-Function Unit is the hardware component of a 3D Pipeline Stage. A FF Unit typically has a unique FF ID associated with it.
Fixed Function	FF	Function of the pipeline that is performed by dedicated (vs. programmable) hardware.
Fixed Function ID	FFID	Unique identifier for a fixed function unit.
FLT_MAX	fmax	The magnitude of the maximum representable single precision floating number according to IEEE-754 standard. FLT_MAX has an exponent of 0xFE and a mantissa of all one's.
Gateway	GW	See Message Gateway.
GEN Core		Alternative name for an EU in the GEN multi-processor system.
General Register File	GRF	Large read/write register file shared by all the EUs for operand sources and destinations. This is the most commonly used read-write register space organized as an array of 256-bit registers for a thread.
General State Base Address	--	The Graphics Address of a block of memory-resident "state data", which includes state blocks, scratch space, constant buffers and kernel programs. The contents of this memory block are referenced via offsets from the contents of the General State Base Address register. See *Graphics Processing Engine*.
Geometry Shader	GS	Fixed-function unit that (if enabled) dispatches "geometry shader" threads on its input primitives. Application-supplied geometry shaders normally expand each input primitive into several output primitives in order to perform 3D modeling algorithms such as fur/fins.
Graphics Address		The GPE virtual address of some memory-resident object. This virtual address gets mapped by a GTT or PGTT to a physical memory address. Note that many memory-resident objects are referenced not with Graphics Addresses, but instead with offsets from a "base address register".
Graphics Processing Engine	GPE	Collective name for the Subsystem, the 3D and Media pipelines, and the Command Streamer.

Term	Abbr.	Definition
Guardband	GB	Region that may be clipped against to make sure objects do not exceed the limitations of the renderer's coordinate space.
Horizontal Stride	HorzStride	The distance in element-sized units between adjacent elements of a GEN region-based GRF access.
Immediate floating point vector	VF	A numerical data type of 32 bits, an immediate floating point vector of type VF contains 4 floating point elements with 8-bit each. The 8-bit floating point element contains a sign field, a 3-bit exponent field and a 4-bit mantissa field. It may be used to specify the type of an immediate operand in an instruction.
Immediate integer vector	V	A numerical data type of 32 bits, an immediate integer vector of type V contains 8 signed integer elements with 4-bit each. The 4-bit integer element is in 2's compliment form. It may be used to specify the type of an immediate operand in an instruction.
Index Buffer	IB	Buffer in memory containing vertex indices.
In-loop Deblocking Filter	ILDB	The deblocking filter operation in the decoding loop. It is a stage after MC in the video decoding pipe.
Instance		In the context of the VF unit, an instance is one of a sequence of sets of similar primitive data. Each set has identical vertex data but may have unique instance data that differentiates it from other sets in the sequence.
Instruction	--	Data in memory directing an EU operation. Instructions are fetched from memory, stored in a cache and executed on one or more GEN cores. Not to be confused with commands which are fetched and parsed by the command streamer and dispatched down the 3D or Media pipeline.
Instruction Pointer	IP	The address (really an offset) of the instruction currently being fetched by an EU. Each EU has its own IP.
Instruction Set Architecture	ISA	The GEN ISA describes the instructions supported by a GEN EU.
Instruction State Cache	ISC	On-chip memory that holds recently-used instructions and state variable values.
Interface Descriptor	--	Media analog of a State Descriptor.
Intermediate Z	IZ	Completion of the Z (depth) test at the front end of the Windower/Masker unit when certain conditions are met (no alpha, no pixel-shader computed Z values, etc.)
Inverse Discrete Cosine Transform	IDCT	the stage in the video decoding pipe between IQ and MC
Inverse Quantization	IQ	A stage in the video decoding pipe between IS and IDCT.
Inverse Scan	IS	A stage in the video decoding pipe between VLD and IQ. In this stage, a sequence of none-zero DCT coefficients are converted into a block (e.g. an 8x8 block) of coefficients. VFE unit has fixed functions to support IS for MPEG-2.
Jitter		Just-in-time compiler.
Kernel	--	A sequence of GEN instructions that is logically part of the driver or generated by the jitter. Differentiated from a Shader which is an application supplied program that is translated by the jitter to GEN instructions.
Least Significant Bit	LSB	
MathBox	--	See Extended Math Unit

Term	Abbr.	Definition
Media	--	Term for operations that are normally performed by the Media pipeline.
Media Pipeline	--	Fixed function stages dedicated to media and “generic” processing, sometimes referred to as the generic pipeline.
Message	--	Messages are data packages transmitted from a thread to another thread, another shared function or another fixed function. Message passing is the primary communication mechanism of GEN architecture.
Message Gateway	--	Shared function that enables thread-to-thread message communication/synchronization used solely by the Media pipeline.
Message Register File	MRF	Write-only registers used by EUs to assemble messages prior to sending and as the operand of a send instruction.
Most Significant Bit	MSB	
Motion Compensation	MC	Part of the video decoding pipe.
Motion Picture Expert Group	MPEG	MPEG is the international standard body JTC1/SC29/WG11 under ISO/IEC that has defined audio and video compression standards such as MPEG-1, MPEG-2, and MPEG-4, etc.
Motion Vector Field Selection	MVFS	A four-bit field selecting reference fields for the motion vectors of the current macroblock.
Multi Render Targets	MRT	Multiple independent surfaces that may be the target of a sequence of 3D or Media commands that use the same surface state.
Normalized Device Coordinates	NDC	Clip Space Coordinates that have been divided by the Clip Space “W” component.
Object	--	A single triangle, line or point.
Open GL	OGL	A Graphics API specification associated with Linux.
Parent Thread	--	A thread corresponding to a root-node or a branch-node in thread generation hierarchy. A parent thread may be a root thread or a child thread depending on its position in the thread generation hierarchy.
Pipeline Stage	--	A abstracted element of the 3D pipeline, providing functions performed by a combination of the corresponding hardware FF unit and the threads spawned by that FF unit.
Pipelined State Pointers	PSP	Pointers to state blocks in memory that are passed down the pipeline.
Pixel Shader	PS	Shader that is supplied by the application, translated by the jitter and is dispatched to the EU by the Windower (conceptually) once per pixel.
Point	--	A drawing object characterized only by position coordinates and width.
Primitive	--	Synonym for object: triangle, rectangle, line or point.
Primitive Topology	--	A composite primitive such as a triangle strip, or line list. Also includes the objects triangle, line and point as degenerate cases.
Provoking Vertex	--	The vertex of a primitive topology from which vertex attributes that are constant across the primitive are taken.
Quad Quad word (QQword)	QQ	A fundamental data type, QQ represents 32 bytes.
Quad Word (QWord)	QW	A fundamental data type, QW represents 8 bytes.

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

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

Term	Abbr.	Definition
Unsigned Byte integer	UB	A numerical data type of 8 bits.
Unsigned Double Word integer	UD	A numerical data type of 32 bits. It may be used to specify the type of an operand in an instruction.
Unsigned Word integer	UW	A numerical data type of 16 bits. It may be used to specify the type of an operand in an instruction.
Unsynchronized Root Thread	--	A root thread that is automatically dispatched by TS.
URB Dereference	--	
URB Entry	UE	URB Entry: A logical entity stored in the URB (such as a vertex), referenced via a URB Handle.
URB Entry Allocation Size	--	Number of URB entries allocated to a Fixed Function unit.
URB Fence	Fence	Virtual, movable boundaries between the URB regions owned by each FF unit.
URB Handle	--	A unique identifier for a URB entry that is passed down a pipeline.
URB Reference	--	
Variable Length Decode	VLD	The first stage of the video decoding pipe that consists mainly of bit-wide operations. GEN supports hardware VLD acceleration in the VFE fixed function stage.
Vertex Buffer	VB	Buffer in memory containing vertex attributes.
Vertex Cache	VC	Cache of Vertex URB Entry (VUE) handles tagged with vertex indices. See the VS chapter for details on this cache.
Vertex Fetcher	VF	The first FF unit in the 3D pipeline responsible for fetching vertex data from memory. Sometimes referred to as the Vertex Formatter.
Vertex Header	--	Vertex data required for every vertex appearing at the beginning of a Vertex URB Entry.
Vertex ID	--	Unique ID for each vertex that can optionally be included in vertex attribute data sent down the pipeline and used by kernel/shader threads.
Vertex Index	--	Offset (in vertex-sized units) of a given vertex in a vertex buffer. Not unique per vertex instance.
Vertex URB Entry	VUE	A URB entry that contains data for a specific vertex.
Vertical Stride	VertStride	The distance in element-sized units between 2 vertically-adjacent elements of a GEN region-based GRF access.
Video Front End	VFE	The first fixed function in the GEN generic pipeline; performs fixed-function media operations.
Viewport	VP	
Windower IZ	WIZ	Term for Windower/Masker that encapsulates its early (“intermediate”) depth test function.
Windower/Masker	WM	Fixed function triangle/line rasterizer.
Word	W	A numerical data type of 16 bits, W represents a signed word integer.

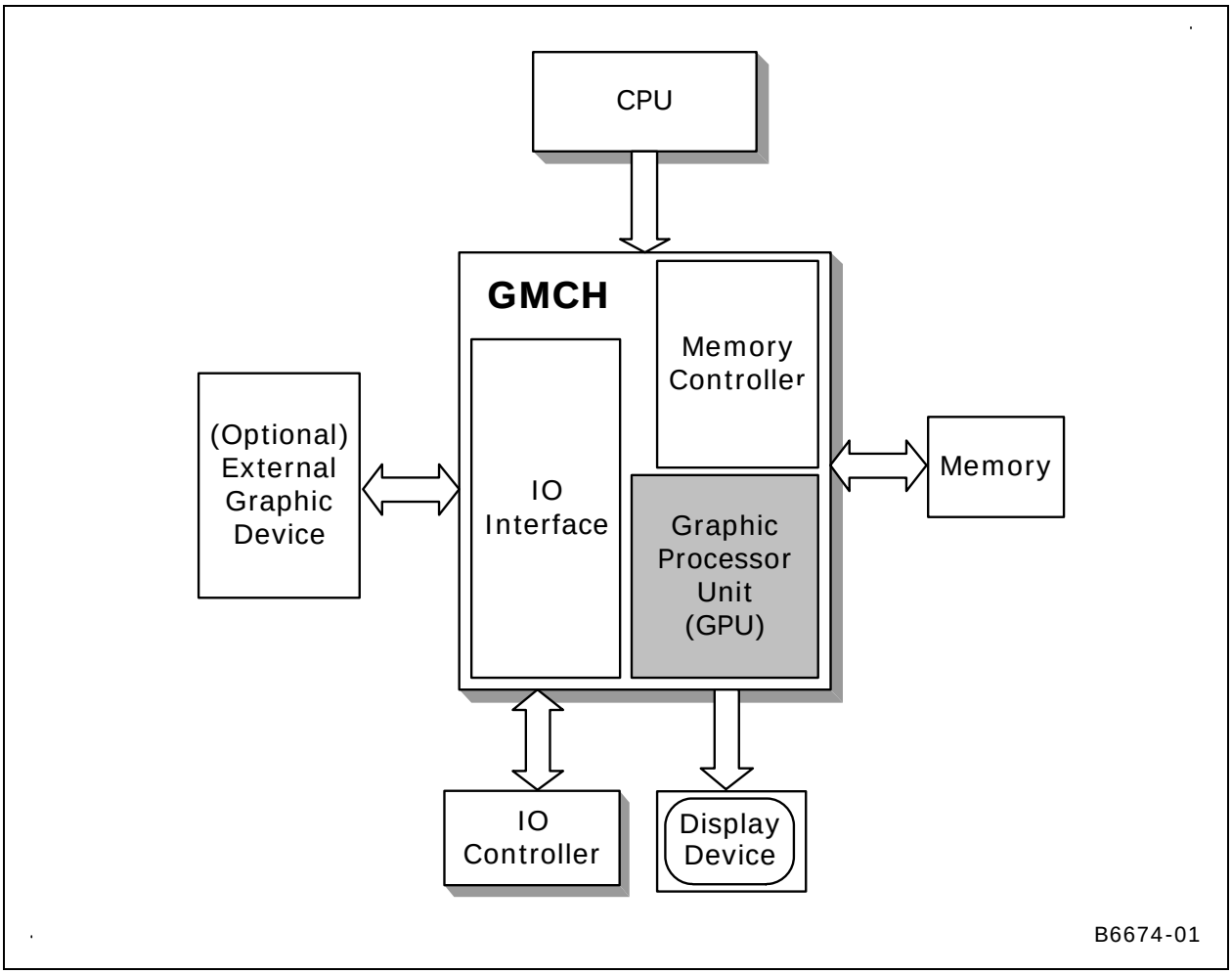
2. Graphics Device Overview

2.1 Graphics Memory Controller Hub (GMCH)

The GMCH is a system memory controller with an integrated graphics device. The integrated graphics device is sometimes referred to in this document as a Graphics Processing Unit (GPU). The GMCH connects to the CPU, via a host bus, and to system memory via a memory bus. The GMCH also contains some IO functionality to interface to an external graphics device and an IO controller. This document will not contain any further references to external graphics devices or IO controllers.

The graphics core, or GPU, resides within the GMCH, which also contains the memory interface, configuration registers, and other chipset functions. The GPU itself is comprised of the command streamer (CS) or command parser, the Memory Interface or MI, the display interface, and (by far the largest element of the GEN family GMCH) the 3D/Media engine. This latter piece is made up of the 3D and media “fixed function” (FF) pipelines, and the GEN subsystem, which these pipelines use to run “shaders” and kernels.

Figure 2-1. GMCH Block Diagram

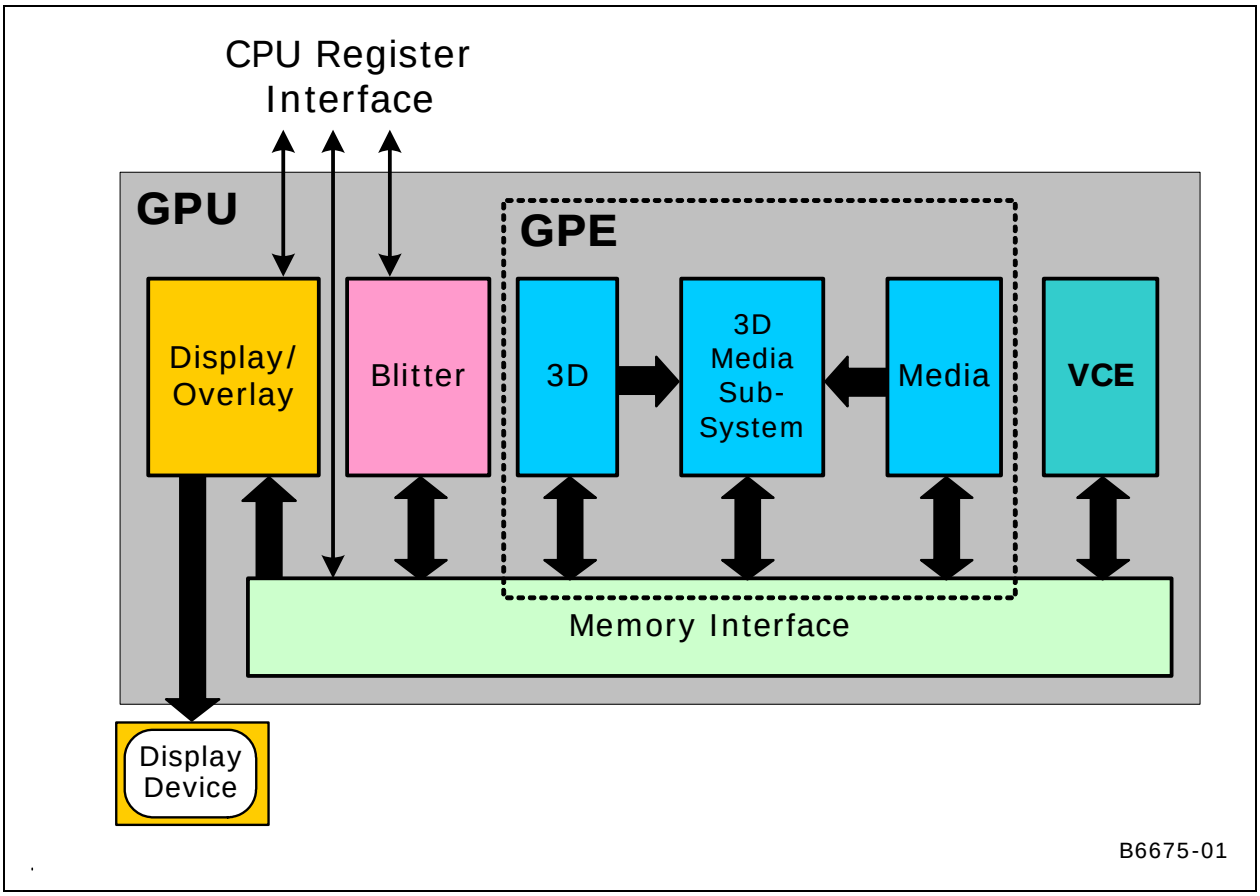


2.2 Graphics Processing Unit (GPU)

The Graphics Processing Unit is controlled by the CPU through a direct interface of memory-mapped IO registers, and indirectly by parsing commands that the CPU has placed in memory. The display interface and blitter (**bl**ock **i**mage **t**ransfer**er**) are controlled primarily by direct CPU register addresses, while the 3D and Media pipelines and the parallel Video Codec Engine (VCE) are controlled primarily through instruction lists in memory.

The GEN subsystem contains an array of cores, or execution units, with a number of “shared functions”, which receive and process messages at the request of programs running on the cores. The shared functions perform critical tasks, such as sampling textures and updating the render target (usually the frame buffer). The cores themselves are described by an instruction set architecture, or ISA.

Figure 2-2. Block Diagram of the GPU



3. Graphics Processing Engine (GPE)

3.1 Introduction

This chapter serves two purposes: It provides a high-level description of the Graphics Processing Engine (GPE) of the Graphics Processing Unit (GPU). It also specifies the programming and behaviors of the functions common to both pipelines (3D, Media) within the GPE. However, details specific to either pipeline are not addressed here.

3.2 Overview

The Graphics Processing Engine (GPE) performs the bulk of the graphics processing provided by the DevSNB GPU. It consists of the 3D and Media fixed-function pipelines, the Command Streamer (CS) unit that feeds them, and the GEN Subsystem that provides the bulk of the computations required by the pipelines.

Figure 3-1. The Graphics Processing Engine

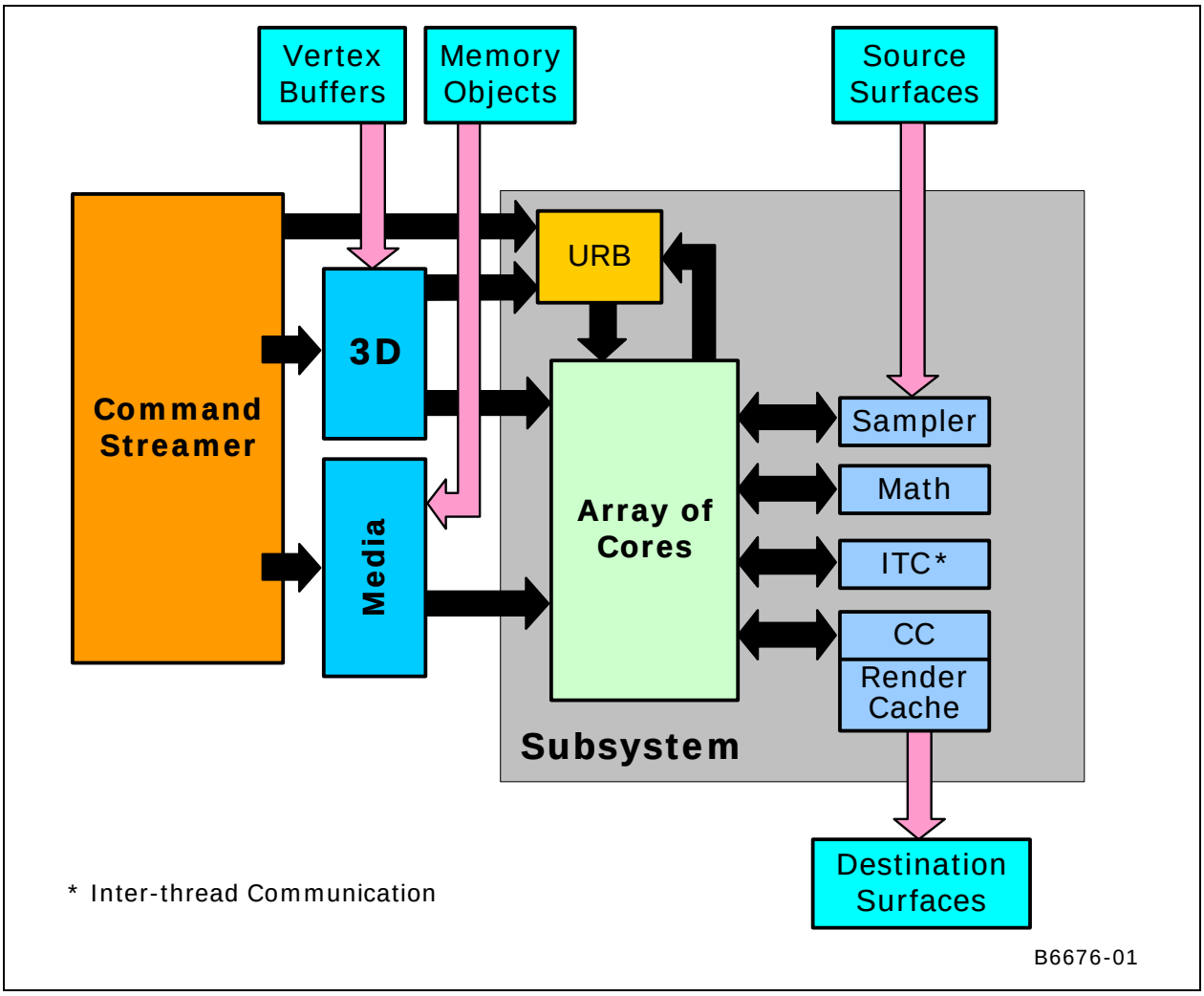
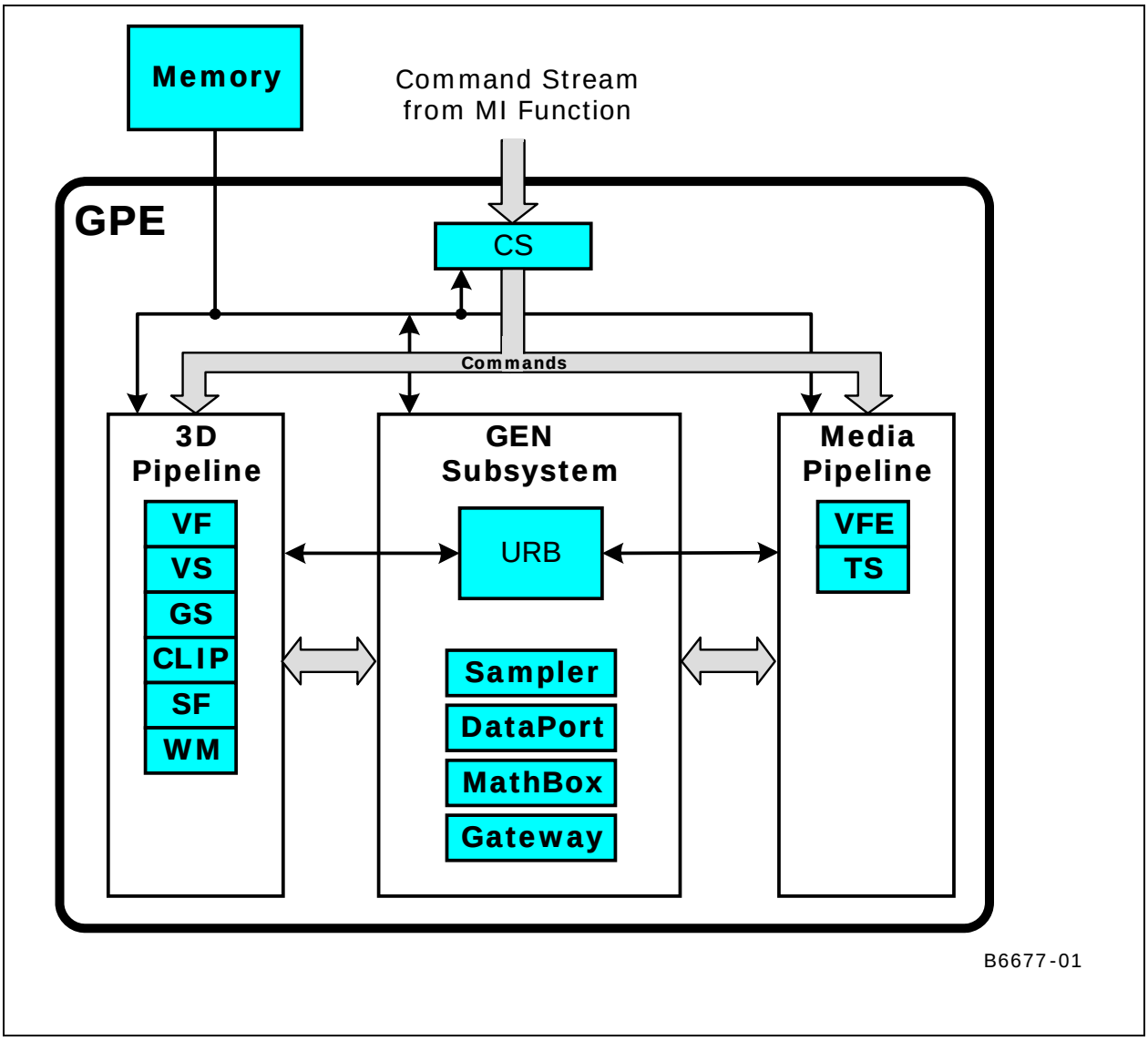


Figure 3-2. GPE Diagram Showing Fixed/Shared Functions



3.2.1 Command Stream (CS) Unit

The Command Stream (CS) unit manages the use of the 3D and Media pipelines; it performs switching between pipelines and forwarding command streams to the currently active pipeline. It manages allocation of the URB and helps support the Constant URB Entry (CURBE) function.

3D Pipeline

The 3D pipeline provides specialized 3D primitive processing functions. These functions are provided by a pipeline of "fixed function" stages (units) and GEN threads spawned by these units. See *3D Pipeline Overview*.

Media Pipeline

The Media pipeline provides both specialized media-related processing functions and the ability to perform more general ("generic") functionality. These Media-specific functions are provided by a Video Front End (VFE) unit. A Thread Spawner (TS) unit is utilized to spawn GEN threads requested by the VFE unit, or as required when the pipeline is used for general processing. See *Media Pipeline Overview*.

Subsystem

The Subsystem is the collective name for the GEN programmable cores, the Shared Functions accessed by them (including the Sampler, Extended Math Unit ("MathBox"), the DataPort, and the Inter-Thread Communication (ITC) Gateway), and the Dispatcher that manages threads running on the cores.

3.2.1.1 Execution Units (EUs)

While the number of EU cores in the GEN subsystem is almost entirely transparent to the programming model, there are a few areas where this parameter comes into play. The amount of scratch space required is a function of (#EUs * #Threads/EU).

Device	# of EUs	#Threads/EU
SNB GT2	12	5
SNB GT1	6	4

3.2.2 GPE Function IDs

The following table lists the assigments (encodings) of the Shared Function and Fixed Function IDs used within the GPE. A Shared Function is a valid target of a message initiated via a 'send' instruction. A Fixed Function is an identifiable unit of the 3D or Media pipeline. Note that the Thread Spawner is both a Shared Function and Fixed Function.

Note: The initial intention was to combine these two ID namespaces, so that (theoretically) an agent (such as the Thread Spawner) that served both as a Shared Function and Fixed Function would have a single, unique 4-bit ID encoding. However, this combination is not a requirement of the architecture.

Table 3-1. Function IDs

ID[3:0]	SFID	Shared Function	FFID	Fixed Function
0x0	SFID_NULL	Null	FFID_NULL	Null
0x1	SFID_MATH	Extended Math	Reserved	---
0x2	SFID_SAMPLER	Sampler	Reserved	---
0x3	SFID_GATEWAY	Message Gateway	Reserved	---
0x4	--	--	Reserved	--
0x5	--	--	Reserved	--
0x6	SFID_URB	URB	Reserved	---
0x7	SFID_SPAWNER	Thread Spawner	FFID_SPAWNER	Thread Spawner

ID[3:0]	SFID	Shared Function	FFID	Fixed Function
0x8	SFID_VME	Video Motion Estimation	FFID_VFE	Video Front End
0xA	Reserved	--	Reserved	--
0xB	Reserved	--	Reserved	--
0xC	Reserved	---	FFID_GS	Geometry Shader
0xD	Reserved	---	FFID_CLIP	Clipper Unit
0xE	Reserved	---	Reserved	--
0xF	Reserved	---	FFID_WM	Windower/Masker Unit

3.3 Pipeline Selection

The PIPELINE_SELECT command is used to specify which GPE pipeline (3D or Media) is to be considered the "current" active pipeline. Issuing 3D-pipeline-specific commands when the Media pipeline is selected, or vice versa, is UNDEFINED.

This command causes the URB deallocation of the previously selected pipe. For example, switching from the 3D pipe to the Media pipe (either within or between contexts) will cause the CS to send a "Deallocating Flush" down the 3D pipe, and each 3D FF will start a URB deallocation sequence after the current tasks are done. Then, the WM will de-reference the current Constant URB Entry, and all 3D URB entries will be deallocated (after some north bus delay) , which allows the CS to set the URB fences for the media pipe. The process relatively is the same for switching from Media to 3D pipes. The deallocating flush goes down the Media pipe, causing each Media function to start a URB deallocation sequence, and the WM will de-reference the current Constant URB entry and all media entries will be de-allocated to allow the CS to set the 3D pipe.

Programming Restriction:

Software must ensure the current pipeline is flushed via an MI_FLUSH or PIPE_CONTROL prior to the execution of PIPELINE_SELECT.

DWord	Bit	Description
0	31:29	**Instruction Type =** GFXPIPE = 3h
	28:16	**3D Instruction Opcode =** PIPELINE_SELECT GFXPIPE[28:27 = 1h, 26:24 = 1h, 23:16 = 04h] (Single DW, Non-pipelined)
	15:2	Reserved: MBZ
	1: 0	**Pipeline Select** 0: 3D pipeline is selected 1: Media pipeline is selected (Includes generic media workloads) 2: Reserved 3: Reserved

The **Pipeline Select** state is contained within the logical context.

Implementation Note: Currently, this bit is only required for switching pipelines. The CS unit needs to know which pipeline (if any) has an outstanding CURBE reference pending. A switch away from that pipeline requires the CS unit to force any CURBE entries to be deallocated.

3.4 URB Allocation

Storage in the URB is divided among the various fixed functions in a programmable fashion using the URB_FENCE command (see below).

Note for [DevSNB+]: URB management is performed for the 3D Pipeline only. Refer to the 3D Pipeline chapter for details.

3.5 Constant URB Entries (CURBEs)

Note for [DevSNB+]: The push constant mechanism is now unit-specific. See the 3D Pipeline chapter for details on push constants for VS, GS, and PS (WM).

3.6 Memory Object Control State

The memory object control state defines behavior of memory accesses beyond the graphics core, including graphics data type that allows selective flushing of data from outer caches, and ability to control cacheability in the outer caches.

This control uses several mechanisms. Control state for all memory accesses can be defined page by page in the GTT entries. Memory objects that are defined by state per surface generally have additional memory object control state in the state structure that defines the other surface attributes. Memory objects without state defining them have memory object state control defined per class in the STATE_BASE_ADDRESS command, with class divisions the same as the base addresses. Finally, some memory objects only have the GTT entry mechanism for defining this control. The table below enumerates the memory objects and location the the control state for each:

Memory Object	Location of Control State
surfaces defined by SURFACE_STATE: sampling engine surfaces, render targets, media surfaces, pull constant buffers	SURFACE_STATE
depth, stencil, and hierarchical depth buffers	corresponding state command that defined the buffer attributes
stateless buffers accessed by data port	STATE_BASE_ADDRESS
indirect state objects	STATE_BASE_ADDRESS
kernel instructions	STATE_BASE_ADDRESS
push constant buffers	3DSTATE_CONSTANT_(VS | GS | PS)
index buffers	3DSTATE_INDEX_BUFFER
vertex buffers	3DSTATE_VERTEX_BUFFERS
indirect media object	STATE_BASE_ADDRESS

Memory Object	Location of Control State
generic state prefetch	GTT control only
ring/batch buffers	GTT control only
context save buffers	GTT control only
store dword	GTT control only

3.6.1 MEMORY_OBJECT_CONTROL_STATE

Bit	Description
3	**Reserved**
2	**Graphics Data Type (GFDT)** This field contains the GFDT bit for this surface when writes occur. GFDT can also be set by the GTT. The effective GFDT is the logical OR of this field with the GFDT from the GTT entry. This field is ignored for reads. The GFDT bit is stored in the LLC and selective cache flushing of lines with GFDT set is supported. It is intended to be set on displayable data, which enables efficient flushing of data to be displayed after rendering, since display engine does not snoop the rendering caches. Note that MLC would need to be completely flushed as it does not allow selective flushing. Format = U1
1:0	**Cacheability Control** This field controls cacheability in the mid-level cache (MLC) and last-level cache (LLC). **[DevSNB]:** The MLC is not implemented on this product, thus data is effectively not cached in the MLC regardless of the setting of this field. Format = U2 enumerated type 00: use cacheability control bits from GTT entry 01: data is not cached in LLC 10: data is cached in LLC 11: Reserved

3.7 Memory Access Indirection

The GPE supports the indirection of certain graphics (GTT-mapped) memory accesses. This support comes in the form of two *base address* state variables used in certain memory address computations with the GPE.

The intent of this functionality is to support the dynamic relocation of certain driver-generated memory structures after command buffers have been generated but prior to the their submittal for execution. For example, as the driver builds the command stream it could append pipeline state descriptors, kernel binaries, etc. to a general state buffer. References to the individual items would be inserting in the command buffers as offsets from the base address of the state buffer. The state buffer could then be freely relocated prior to command buffer execution, with the driver only needing to specify the final base address of the state buffer. Two base addresses are provided to permit surface-related state (binding tables, surface state tables) to be maintained in a state buffer separate from the general state buffer.

While the use of these base addresses is unconditional, the indirection can be effectively disabled by setting the base addresses to zero. The following table lists the various GPE memory access paths and which base address (if any) is relevant.

Table 3-2. Base Address Utilization

Base Address Used	Memory Accesses
General State Base Address	DataPort memory accesses resulting from **'stateless' DataPort Read/Write requests**. See *DataPort* for a definition of the 'stateless' form of requests.
Dynamic State Base Address	Sampler reads of SAMPLER_STATE data and associated SAMPLER_BORDER_COLOR_STATE.
	Viewport states used by CLIP, SF, and WM/CC
	COLOR_CALC_STATE, DEPTH_STENCIL_STATE, and BLEND_STATE
	Push Constants (depending on state of **INSTPM<CONSTANT_BUFFER Address Offset Disable>**)
Instruction Base Address	**Normal EU instruction stream** (non-system routine)
	System routine EU instruction stream (starting address = SIP)
Surface State Base Address	Sampler and DataPort reads of BINDING_TABLE_STATE, as referenced by BT pointers passed via 3DSTATE_BINDING_TABLE_POINTERS
	Sampler and DataPort reads of SURFACE_STATE data
Indirect Object Base Address	**MEDIA_OBJECT Indirect Data** accessed by the CS unit .
None	CS unit reads from **Ring Buffers**, **Batch Buffers**
	CS writes resulting from PIPE_CONTROL command
	All VF unit memory accesses (**Index Buffers**, **Vertex Buffers**)
	All Sampler **Surface Memory Data** accesses (texture fetch, etc.)

Base Address Used	Memory Accesses
	All **DataPort memory accesses** except 'stateless' DataPort Read/Write requests (e.g., RT accesses.) See *Data Port* for a definition of the 'stateless' form of requests.
	Memory reads resulting from **STATE_PREFETCH** commands
	Any **physical memory access** by the device
	GTT-mapped accesses not included above (i.e., default)
	Push Constants (depeding on state of **INSTPM<CONSTANT_BUFFER Address Offset Disable>**)

The following notation is used in the BSpec to distinguish between addresses and offsets:

Notation	Definition
PhysicalAddress[n:m]	Corresponding bits of a physical graphics memory byte address (not mapped by a GTT)
GraphicsAddress[n:m]	Corresponding bits of an absolute, virtual graphics memory byte address (mapped by a GTT)
GeneralStateOffset[n:m]	Corresponding bits of a relative byte offset added to the General State Base Address value, the result of which is interpreted as a virtual graphics memory byte address (mapped by a GTT)
DynamicStateOffset[n:m]	Corresponding bits of a relative byte offset added to the Dynamic State Base Address value, the result of which is interpreted as a virtual graphics memory byte address (mapped by a GTT)
InstructionBaseOffset[n:m]	Corresponding bits of a relative byte offset added to the Instruction Base Address value, the result of which is interpreted as a virtual graphics memory byte address (mapped by a GTT)
SurfaceStateOffset[n:m]	Corresponding bits of a relative byte offset added to the Surface State Base Address value, the result of which is interpreted as a virtual graphics memory byte address (mapped by a GTT)

3.7.1 STATE_BASE_ADDRESS

The STATE_BASE_ADDRESS command sets the base pointers for subsequent state, instruction, and media indirect object accesses by the GPE. (See Table 3-2. Base Address Utilization for details)

Programming Notes:

- The following commands must be reissued following any change to the base addresses:
    - 3DSTATE_PIPELINE_POINTERS
    - 3DSTATE_BINDING_TABLE_POINTERS
    - MEDIA_STATE_POINTERS.
- Execution of this command causes a full pipeline flush, thus its use should be minimized for higher performance.

3.7.1.1 STATE_BASE_ADDRESS [DevSNB]

STATE_BASE_ADDRESS

Project:	[DevSNB+]	**Length Bias:**	2

The STATE_BASE_ADDRESS command sets the base pointers for subsequent state, instruction, and media indirect object accesses by the GPE. (See Table 3-2. Base Address Utilization for details)

Programming Notes:

- The following commands must be reissued following any change to the base addresses:
  - 3DSTATE_CC_POINTERS
  - 3DSTATE_BINDING_TABLE_POINTERS
  - 3DSTATE_SAMPLER_STATE_POINTERS
  - 3DSTATE_VIEWPORT_STATE_POINTERS
  - MEDIA_STATE_POINTERS
- Execution of this command causes a full pipeline flush, thus its use should be minimized for higher performance.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 0h GFXPIPE_COMMON Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h GFXPIPE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 01h STATE_BASE_ADDRESS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 8h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All

STATE_BASE_ADDRESS

1	31:12	**General State Base Address** Project: All Format: GraphicsAddress[31:12] Specifies the 4K-byte aligned base address for general state accesses. See Table 3-2 for details on where this base address is used.
	11:8	**General State Memory Object Control State** Project: All Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for indirect state using the **General State Base Address**, with the exception of the stateless data port accesses.
	7:4	**Stateless Data Port Access Memory Object Control State** Project: All Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for stateless data port accesses.
	3	**Stateless Data Port Access Force Write Thru** Project: All Format: U1 0: If the stateless data port access memory object control indicates L3 cachable the accesses will be write back cacheable. 1: If the stateless data port access memory object control indicates L3 cachable the accesses will be write thru cacheable.
	2:1	**Reserved** Project: All Format: MBZ
	0	**General State Base Address Modify Enable** Project: All Format: Enable The other fields in this dword are updated only when this bit is set. **Value** / **Name** / **Description** / **Project** 0h / Disable / Ignore the updated address / All 1h / Enable / Modify the address / All
2	31:12	**Surface State Base Address** Project: All Format: GraphicsAddress[31:12] Specifies the 4K-byte aligned base address for binding table and surface state accesses. See Table 3-2 for details on where this base address is used.
	11:8	**Surface State Memory Object Control State** Project: All Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for indirect state using the **Surface State Base Address**.
	7:1	**Reserved** Project: All Format: MBZ

STATE_BASE_ADDRESS

	0	**Surface State Base Address Modify Enable** Project: All Format: Enable The other fields in this dword are updated only when this bit is set. **Value** / **Name** / **Description** / **Project** 0h / Disable / Ignore the updated address / All 1h / Enable / Modify the address / All
3	31:12	**Dynamic State Base Address** Project: All Format: GraphicsAddress[31:12] Specifies the 4K-byte aligned base address for sampler and viewport state accesses. See Table 3-2 for details on where this base address is used.
	11:8	**Dynamic State Memory Object Control State** Project: All Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for indirect state using the **Dynamic State Base Address**. Push constants defined in 3DSTATE_CONSTANT_(VS | GS | PS) commands do not use this control state, although they can use the corresponding base address. The memory object control state for push constants is defined within the command.
	7:1	**Reserved** Project: All Format: MBZ
	0	**Dynamic State Base Address Modify Enable** Project: All Format: Enable The other fields in this dword are updated only when this bit is set. **Value** / **Name** / **Description** / **Project** 0h / Disable / Ignore the updated address / All 1h / Enable / Modify the address / All
4	31:12	**Indirect Object Base Address** Project: All Format: GraphicsAddress[31:12] Specifies the 4K-byte aligned base address for indirect object load in MEDIA_OBJECT command. See Table 3-2 for details on where this base address is used.
	11:8	**Indirect Object Memory Object Control State** Project: All Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for indirect objects using the **Indirect Object Base Address**.
	7:1	**Reserved** Project: All Format: MBZ

STATE_BASE_ADDRESS

	0	**Indirect Object Base Address Modify Enable** Project: All Format: Enable The other fields in this dword are updated only when this bit is set. **Value** / **Name** / **Description** / **Project** 0h / Disable / Ignore the updated address / All 1h / Enable / Modify the address / All
5	31:12	**Instruction Base Address** Project: All Format: GraphicsAddress[31:12] Specifies the 4K-byte aligned base address for all EU instruction accesses.
	11:8	**Instruction Memory Object Control State** Project: All Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for EU instructions using the **Instruction Base Address**.
	7:1	**Reserved** Project: All Format: MBZ
	0	**Instruction Base Address Modify Enable** Project: All Format: Enable The other fields in this dword are updated only when this bit is set. **Value** / **Name** / **Description** / **Project** 0h / Disable / Ignore the updated address / All 1h / Enable / Modify the address / All
6	31:12	**General State Access Upper Bound** Project: All Format: GraphicsAddress[31:12] Specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address for general state accesses. This includes all accesses that are offset from **General State Base Address** (see Table 3-2). Read accesses from this address and beyond will return UNDEFINED values. Data port writes to this address and beyond will be “dropped on the floor” (all data channels will be disabled so no writes occur). Setting this field to 0 will cause this range check to be ignored. If non-zero, this address must be greater than the **General State Base Address**.
	11:1	**Reserved** Project: All Format: MBZ

STATE_BASE_ADDRESS

	0	**General State Access Upper Bound Modify Enable** Project: All Format: Enable The bound in this dword is updated only when this bit is set. **Value** / **Name** / **Description** / **Project** 0h / Disable / Ignore the updated bound / All 1h / Enable / Modify the bound / All
7	31:12	**Dynamic State Access Upper Bound** Project: All Format: GraphicsAddress[31:12] Specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address for dynamic state accesses. This includes all accesses that are offset from **Dynamic State Base Address** (see Table 3-2). Read accesses from this address and beyond will return UNDEFINED values. Data port writes to this address and beyond will be "dropped on the floor" (all data channels will be disabled so no writes occur). Setting this field to 0 will cause this range check to be ignored. If non-zero, this address must be greater than the **Dynamic State Base Address**.
	11:1	**Reserved** Project: All Format: MBZ
	0	**Dynamic State Access Upper Bound Modify Enable** Project: All Format: Enable The bound in this dword is updated only when this bit is set. **Value** / **Name** / **Description** / **Project** 0h / Disable / Ignore the updated bound / All 1h / Enable / Modify the bound / All
8	31:12	**Indirect Object Access Upper Bound** Project: All Format: GraphicsAddress[31:12] This field specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address access by an indirect object load in a MEDIA_OBJECT command. Indirect data accessed at this address and beyond will appear to be 0. Setting this field to 0 will cause this range check to be ignored. If non-zero, this address must be greater than the **Indirect Object Base Address**. Hardware ignores this field if indirect data is not present. Setting this field to FFFFFh will cause this range check to be ignored.
	11:1	**Reserved** Project: All Format: MBZ

STATE_BASE_ADDRESS		
	0	**Indirect Object Access Upper Bound Modify Enable** Project: All Format: Enable The bound in this dword is updated only when this bit is set. **Value** / **Name** / **Description** / **Project** 0h / Disable / Ignore the updated bound / All 1h / Enable / Modify the bound / All
9	31:12	**Instruction Access Upper Bound** Project: All Format: GraphicsAddress[31:12] This field specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address access by an EU instruction. Instruction data accessed at this address and beyond will return UNDEFINED values. Setting this field to 0 will cause this range check to be ignored. If non-zero, this address must be greater than the **Instruction Base Address**.
	11:1	**Reserved** Project: All Format: MBZ
	0	**Instruction Access Upper Bound Modify Enable** Project: All Format: Enable The bound in this dword is updated only when this bit is set. **Value** / **Name** / **Description** / **Project** 0h / Disable / Ignore the updated bound / All 1h / Enable / Modify the bound / All

3.8 Instruction and State Prefetch

The STATE_PREFETCH command is provided strictly as an optional mechanism to possibly enhance pipeline performance by prefetching data into the GPE's Instruction and State Cache (ISC).

3.8.1 STATE_PREFETCH

STATE_PREFETCH			
Project:	All	**Length Bias:**	2

(*This command is provided strictly for performance optimization opportunities, and likely requires some experimentation to evaluate the overall impact of additional prefetching.*)

The STATE_PREFETCH command causes the GPE to attempt to prefetch a sequence of 64-byte cache lines into the GPE-internal cache ("L2 ISC") used to access EU kernel instructions and fixed/shared function indirect state data. While state descriptors, surface state, and sampler state are automatically prefetched by the GPE, this command may be used to prefetch data not automatically prefetched, such as: 3D viewport state; Media pipeline Interface Descriptors; EU kernel instructions.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 0h GFXPIPE_COMMON Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h GFXPIPE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 03h STATE_PREFETCH Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:6	**Prefetch Pointer** Project: All Format: GraphicsAddress[31:6] Specifies the 64-byte aligned address to start the prefetch from. This pointer is an absolute virtual address, it is *not* relative to any base pointer.
	5:3	**Reserved** Project: All Format: MBZ
	2:0	**Prefetch Count** Project: All Format: U3 count of cache lines (minus one) Range [0,7] indicating a count of [1,8] Indicates the number of contiguous 64-byte cache lines that will be prefetched.

3.9 System Thread Configuration

3.9.1 STATE_SIP

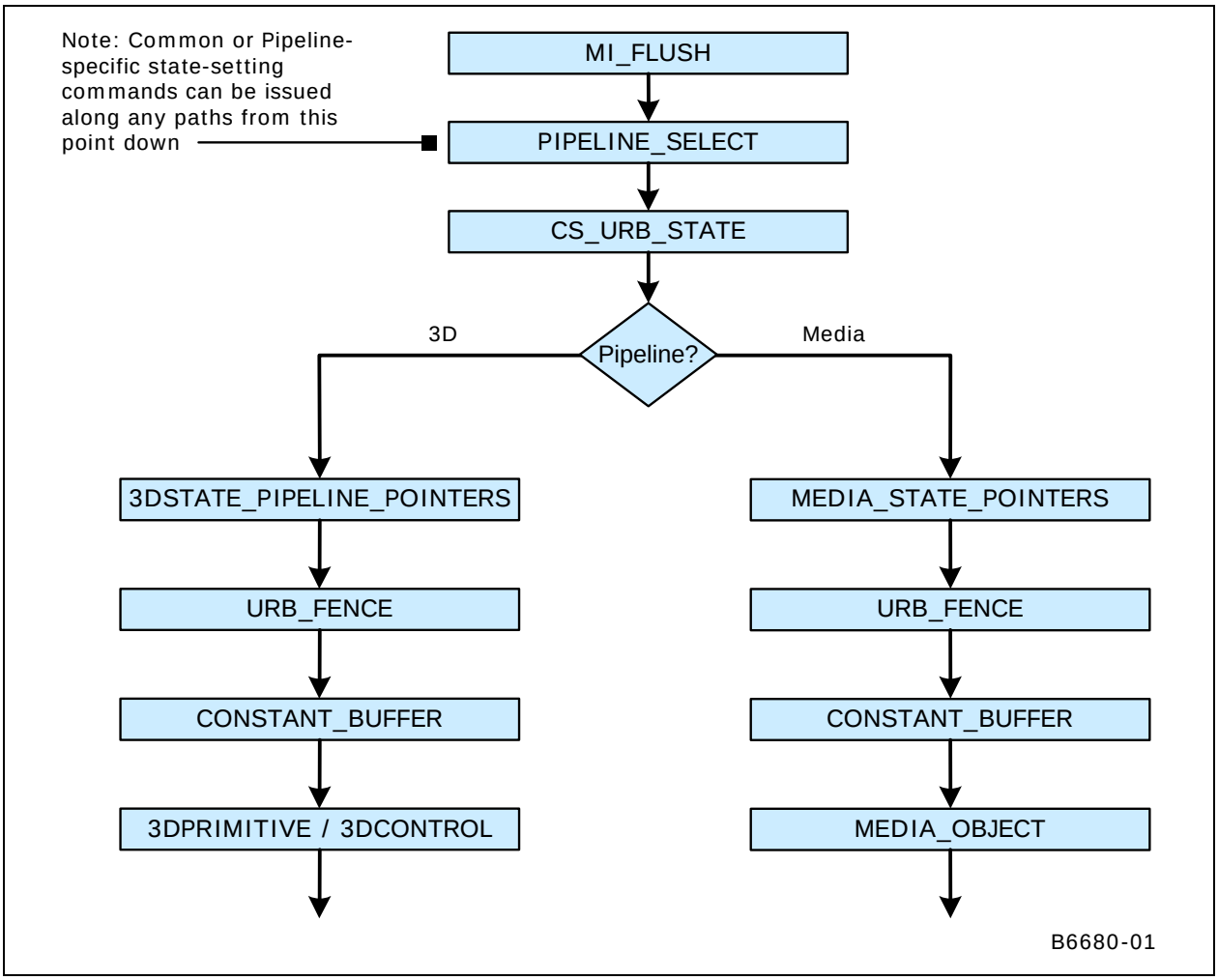
STATE_SIP			
Project:	All	**Length Bias:**	2
The STATE_SIP command specifies the starting instruction location of the System Routine that is shared by all threads in execution.			

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 0h GFXPIPE_COMMON Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h GFXPIPE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 02h STATE_SIP Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:4	**System Instruction Pointer (SIP)** Project: All Format: InstructionBaseOffset[31:4] Specifies the instruction address of the system routine associated with the current context as a 128-bit granular offset from the **Instruction Base Address**. SIP is shared by all threads in execution. The address specifies the double quadword aligned instruction location.
	3:0	**Reserved** Project: All Format: MBZ

3.10 Command Ordering Rules

There are several restrictions regarding the ordering of commands issued to the GPE. This subsection describes these restrictions along with some explanation of why they exist. Refer to the various command descriptions for additional information.

The following flowchart illustrates an example ordering of commands which can be used to perform activity within the GPE.



3.10.1 PIPELINE_SELECT

The previously-active pipeline needs to be flushed via the MI_FLUSH command immediately before switching to a different pipeline via use of the PIPELINE_SELECT command. Refer to Section 0 for details on the PIPELINE_SELECT command.

3.10.2 PIPE_CONTROL

The PIPE_CONTROL command does not require URB fencing/allocation to have been performed, nor does it rely on any other pipeline state. It is intended to be used on both the 3D pipe and the Media pipe. It has special optimizations to support the pipelining capability in the 3D pipe which do not apply to the Media pipe.

3.10.3 URB-Related State-Setting Commands

Several commands are used (among other things) to set state variables used in URB entry allocation --- specifically, the **Number of URB Entrie**s and the **URB Entry Allocation Size** state variables associated with various pipeline units. These state variables must be set-up prior to the issuing of a URB_FENCE command. (See the subsection on URB_FENCE below).

CS_URB_STATE (only) specifies these state variables for the common CS FF unit. 3DSTATE_PIPELINED_POINTERs sets the state variables for FF units in the 3D pipeline, and MEDIA_STATE_POINTERS sets them for the Media pipeline. Depending on which pipeline is currently active, only one of these commands needs to be used. Note that these commands can also be reissued at a later time to change other state variables, though if a change is made to (a) any **Number of URB Entrie**s and the **URB Entry Allocation Size** state variables or (b) the **Maximum Number of Threads** state for the GS or CLIP FF units, a URB_FENCE command must follow.

3.10.4 Common Pipeline State-Setting Commands

The following commands are used to set state common to both the 3D and Media pipelines. This state is comprised of CS FF unit state, non-pipelined global state (EU, etc.), and Sampler shared-function state.

- STATE_BASE_ADDRESS
- STATE_SIP
- 3DSTATE_SAMPLER_PALETTE_LOAD
- 3DSTATE_CHROMA_KEY

The state variables associated with these commands must be set appropriately prior to initiating activity within a pipeline (i.e., 3DPRIMITIVE or MEDIA_OBJECT).

3.10.5 3D Pipeline-Specific State-Setting Commands

The following commands are used to set state specific to the 3D pipeline.

- 3DSTATE_PIPELINED_POINTERS
- 3DSTATE_BINDING_TABLE_POINTERS
- 3DSTATE_VERTEX_BUFFERS
- 3DSTATE_VERTEX_ELEMENTS
- 3DSTATE_INDEX_BUFFERS
- 3DSTATE_VF_STATISTICS
- 3DSTATE_DRAWING_RECTANGLE

- 3DSTATE_CONSTANT_COLOR
- 3DSTATE_DEPTH_BUFFER
- 3DSTATE_POLY_STIPPLE_OFFSET
- 3DSTATE_POLY_STIPPLE_PATTERN
- 3DSTATE_LINE_STIPPLE
- 3DSTATE_GLOBAL_DEPTH_OFFSET

The state variables associated with these commands must be set appropriately prior to issuing 3DPRIMITIVE.

3.10.6 Media Pipeline-Specific State-Setting Commands

The following commands are used to set state specific to the Media pipeline.

- MEDIA_STATE_POINTERS

The state variables associated with this command must be set appropriately prior to issuing MEDIA_OBJECT.

3.10.7 URB_FENCE (URB Fencing & Entry Allocation)

URB_FENCE command is used to initiate URB entry deallocation/allocation processes within pipeline FF units. The URB_FENCE command is first processed by the CS FF unit, and is then directed down the currently selected pipeline to the FF units comprising that pipeline.

As the FF units receive the URB_FENCE command, a URB entry deallocation/allocation process with be initiated if (a) the FF unit is currently enabled (note that some cannot be disabled) and (b) the **ModifyEnable** bit associated with that FF unit's **Fence** value is set. If these conditions are met, the deallocation of the FF unit's currently-allocated URB entries (if any) commences. (Implementation Note: For better performance, this deallocation proceeds in parallel with allocation of new handles).

Modifying the CS URB allocation via URB_FENCE invalidates any previous CURBE entries. Therefore software must subsequently [re]issue a CONSTANT_BUFFER command before CURBE data can be used in the pipeline.

The allocation of new handles (if any) for the FF unit then commences. The parameters used to perform this allocation come from (a) the URB_FENCE **Fence** values, and (b) the relevant URB entry state associated with the FF unit: specifically, the **Number of URB Entrie**s and the **URB Entry Allocation Size**. For the CS unit, this state is programmed via CS_URB_STATE, while the other FF units receive this state indirectly via PIPELINED_STATE_POINTERS or MEDIA_STATE_POINTERS commands.

Although a FF unit's allocation process relies on it's URB **Fence** as well as the relevant FF unit pipelined state, only the URB_FENCE command initiates URB entry deallocation/allocation. This imposes the following restriction: <u>If a change is made to (a) the **Number of URB Entries** or **URB Entry Allocation Size** state for a given FF unit or (b) the **Maximum Number of Threads** state for the GS or CLIP FF units,</u>

a URB_FENCE command specifying a valid URB Fence state for that FF unit must be subsequently issued – at some point prior to the next CONSTANT_BUFFER, 3DPRIMITIVE (if using the 3D pipeline) or MEDIA_OBJECT (if using the Media pipeline). It is invalid to change **Number of URB Entries** or **URB Entry Allocation Size** state for an enabled FF units without also issuing a subsequent URB_FENCE command specifying a valid **Fence** valid for that FF unit.

It is valid to change a FF unit's Fence value without specifying a change to its **Number of URB Entries** or **URB Entry Allocation Size** state, though the values must be self-consistent.

3.10.8 CONSTANT_BUFFER (CURBE Load)

The CONSTANT_BUFFER command is used to load constant data into the CURBE URB entries owned by the CS unit. In order to write into the URB, CS URB fencing and allocation must have been established. Therefore, CONSTANT_BUFFER can only be issued after CS_URB_STATE and URB_FENCE commands have been issued, and prior to any other pipeline processing (i.e., 3DPRIMITIVE or MEDIA_OBJECT). See the definition of CONSTANT_BUFFER for more details.

Modifying the CS URB allocation via URB_FENCE invalidates any previous CURBE entries. Therefore software must subsequently [re]issue a CONSTANT_BUFFER command before CURBE data can be used in the pipeline.

3.10.9 3DPRIMITIVE

Before issuing a 3DPRIMITIVE command, all state (with the exception of MEDIA_STATE_POINTERS) needs to be valid. Therefore the commands used to set this state need to have been issued at some point prior to the issue of 3DPRIMITIVE.

3.10.10MEDIA_OBJECT

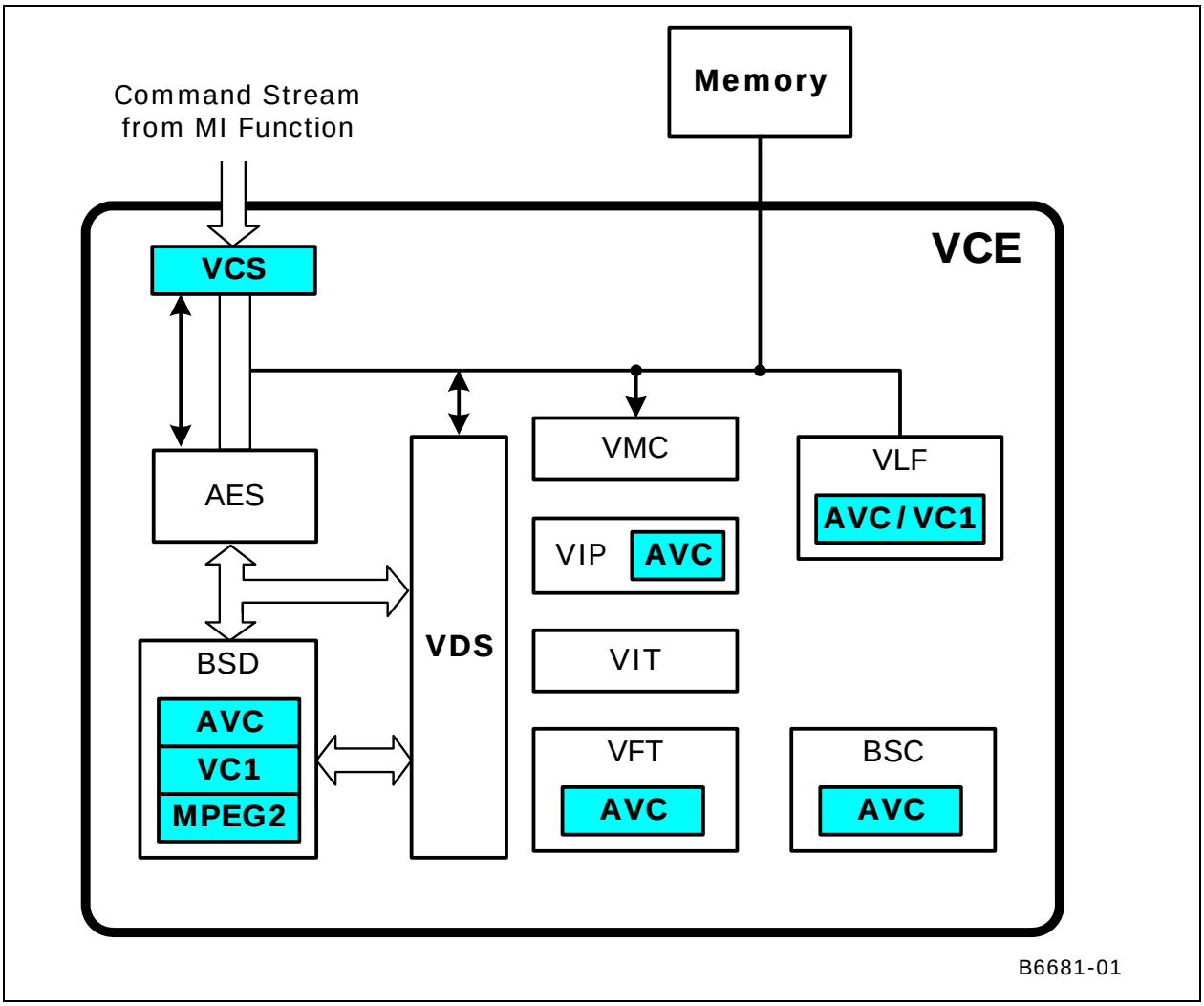
Before issuing a MEDIA_OBJECT command, all state (with the exception of 3D-pipeline-specific state) needs to be valid. Therefore the commands used to set this state need to have been issued at some point prior to the issue of MEDIA_OBJECT.

4. Video Codec Engine

The parallel Video Codec Engine (VCE) is a fixed function video decoder and encoder engine. It is also referred to as the multi-format codec (MFX) engine, as a unified fixed function pipeline is implemented to support multiple video coding standards such as MPEG2, VC1 and AVC.

- VCS – VCE Command Streamer unit (also referred to as BCS)
- BSD – Bitstream Decoder unit
- VDS – Video Dispatcher unit
- VMC – Video Motion Compensation unit
- VIP – Video Intra Prediction unit
- VIT – Video Inverse Transform unit
- VLF – Video Loop Filter unit
- VFT – Video Forward Transform unit (encoder only)
- BSC – Bitstream Encoder unit (encoder only)

Figure 4-1. VCE Diagram



Device	AVC BSD	VC1 BSD	AVC Dec	VC1 Dec	MPEG2 Dec	AVC Enc
[DevSNB]	No	No	Yes	Yes	Yes	Yes

4.1 Video Command Streamer (VCS)

VCS (Video Command Streamer) unit is primarily served as the software programming interface between the O/S driver and the MFD Engine. It is responsible for fetching, decoding, and dispatching of data packets (Media Commands with the header DW removed) to the front end interface module of MFX Engine.

Its logic functions include

- MMIO register programming interface

- DMA action for fetching of ring data from memory
- Management of the Head pointer for the Ring Buffer
- Decode of ring data and sending it to the appropriate destination; AVC, VC1 or MPEG2 engine
- Handling of user interrupts and ring context switch interrupt
- Flushing the MFX Engine
- Handle NOP

The register programming (RM) bus is a dword interface bus that is driven by the Gx Command Streamer. The VCS unit will only claim memory mapped I/O cycles that are targeted to its range of 0x4000 to 0x4FFFF. The Gx and MFX Engines use semaphore to synchronize their operations.

Any interaction and control protocols between the VCS and Gx CS in IronLake will remain the same as in Cantiga. But in Gesher, VCS will operate completely independent of the Gx CS.

The simple sequence of events is as follows: a ring (say PRB0) is programmed by a memory-mapped register write cycle. The DMA inside VCS is kicked off. The DMA fetches commands from memory based on the starting address and head pointer. The DMA requests cache lines from memory (one cacheline CL at a time). There is guaranteed space in the DMA FIFO (16 CL deep) for data coming back from memory. The DMA control logic has copies of the head pointer and the tail pointer. The DMA increments the head pointer after making requests for ring commands. Once the DMA copy of the head pointer becomes equal to the tail pointer, the DMA stops requesting.

The parser starts executing once the DMA FIFO has valid commands. All the commands have a header dword packet. Based on the encoding in the header packet, the command may be targeted towards AVC/VC1/MPEG2 engine or the command parser. After execution of every command, the actual head pointer is updated. The ring is considered empty when the head pointer becomes equal to the tail pointer.

5. Graphics Command Formats

5.1 Command Formats

This section describes the general format of the graphics device commands.

Graphics commands are defined with various formats. The first DWord of all commands is called the *header* DWord. The header contains the only field common to all commands -- the *client* field that determines the device unit that will process the command data. The Command Parser examines the client field of each command to condition the further processing of the command and route the command data accordingly.

Some GEN Devices include two Command Parsers, each controlling an independent processing engine. These will be referred to in this document as the Render Command Parser (RCP) and the Video Codec Command Parser (VCCP).

Valid client values for the Render Command Parser are:

Client #	Client
0	Memory Interface (MI_xxx)
1	Miscellaneous
2	2D Rendering (xxx_BLT_xxx)
3	Graphics Pipeline (3D and Media)
4-7	Reserved

Valid client values for the Video Codec Command Parser are:

Client #	Client
0	Memory Interface (MI_xxx)
1-2	Reserved
3	AVC and VC1 State and Object Commands
4-7	Reserved

Graphics commands vary in length, though are always multiples of DWords. The length of a command is either:

- Implied by the client/opcode
- Fixed by the client/opcode yet included in a header field (so the Command Parser explicitly knows how much data to copy/process)
- Variable, with a field in the header indicating the total length of the command

Note that command *sequences* require QWord alignment and padding to QWord length to be placed in Ring and Batch Buffers.

The following subsections provide a brief overview of the graphics commands by client type provides a diagram of the formats of the header DWords for all commands. Following that is a list of command mnemonics by client type.

5.1.1 Memory Interface Commands

Memory Interface (MI) commands are basically those commands which do not require processing by the 2D or 3D Rendering/Mapping engines. The functions performed by these commands include:

Control of the command stream (e.g., Batch Buffer commands, breakpoints, ARB On/Off, etc.)

Hardware synchronization (e.g., flush, wait-for-event)

Software synchronization (e.g., Store DWORD, report head)

Graphics buffer definition (e.g., Display buffer, Overlay buffer)

Miscellaneous functions

Refer to the *Memory Interface Commands* chapter for a description of these commands.

5.1.2 2D Commands

The 2D commands include various flavors of Blt operations, along with commands to set up Blt engine state without actually performing a Blt. Most commands are of fixed length, though there are a few commands that include a variable amount of "inline" data at the end of the command.

Refer to the *2D Commands* chapter for a description of these commands.

5.1.3 3D/Media Commands

The 3D/Media commands are used to program the graphics pipelines for 3D or media operations.

Refer to the *3D* chapter for a description of the 3D state and primitive commands and the *Media* chapter for a description of the media-related state and object commands.

5.1.4 Video Codec Commands

5.1.4.1 MFX Commands [DevSNB+]

The MFX commands are used to program the multi-format codec engine attached to the Video Codec Command Parser. See the *MFD and MFC* chapters for a description of these commands.

Table 5-1. RCP Command Header Format

Bits					
TYPE	**31:29**	**28:24**	**23**	**22**	**21:0**
Memory Interface (MI)	000	Opcode 00h – NOP 0Xh – Single DWord Commands 1Xh – Two+ DWord Commands 2Xh – Store Data Commands 3Xh – Ring/Batch Buffer Cmds		Identification No./DWord Count Command Dependent Data 5:0 – DWord Count 5:0 – DWord Count 5:0 – DWord Count	
Reserved	001	Opcode – 11111	23:19 Sub Opcode 00h – 01h	18:16 Re-served	15:0 DWord Count
2D	010	Opcode			Command Dependent Data 4:0 – DWord Count

TYPE	**31:29**	**28:27**	**26:24**	**23:16**	**15:8**	**7:0**
Common	011	00	Opcode – 000	Sub Opcode	Data	DWord Count
Common (NP)	011	00	Opcode – 001	Sub Opcode	Data	DWord Count
Reserved	011	00	Opcode – 010 – 111			
Single Dword Command	011	01	Opcode – 000 – 001	Sub Opcode		N/A
Reserved	011	01	Opcode – 010 – 111			
Media State	011	10	Opcode – 000	Sub Opcode		Dword Count
Media Object	011	10	Opcode – 001 – 010	Sub Opcode	Dword Count	
Reserved	011	10	Opcode – 011 – 111			
3DState	011	11	Opcode – 000	Sub Opcode	Data	DWord Count
3DState (NP)	011	11	Opcode – 001	Sub Opcode	Data	DWord Count
PIPE_Control	011	11	Opcode – 010		Data	DWord Count
3DPrimitive	011	11	Opcode – 011		Data	DWord Count
Reserved	011	11	Opcode – 100 – 111			
Reserved	1XX	XX				

NOTES:

1. The qualifier "NP" indicates that the state variable is non-pipelined and the render pipe is flushed before such a state variable is updated. The other state variables are pipelined (default).

Table 5-2. VCCP Command Header Format

Bits					
TYPE	**31:29**	**28:24**	**23**	**22**	**21:0**
Memory Interface (MI)	000	Opcode 00h – NOP 0Xh – Single DWord Commands 1Xh – Reserved 2Xh – Store Data Commands 3Xh – Ring/Batch Buffer Cmds		Identification No./DWord Count Command Dependent Data 5:0 – DWord Count 5:0 – DWord Count 5:0 – DWord Count	

TYPE	**31:29**	**28:27**	**26:24**	**23:16**	**15:0**
Reserved	011	00	XXX	XX	
MFX Single DW	011	01	000	Opcode: 0h	0
Reserved	011	01	1XX		
Reserved	011	10	0XX		
AVC State	011	10	100	Opcode: 0h – 4h	DWord Count
AVC Object	011	10	100	Opcode: 8h	DWord Count
VC1 State	011	10	101	Opcode: 0h – 4h	DWord Count
VC1 Object	011	10	101	Opcode: 8h	DWord Count
Reserved	011	10	110		
Reserved	011	10	110		
Reserved	011	10	11X		
Reserved	011	11	XXX		

TYPE	**31:29**	**28:27**	**26:24**	**23:21**	**20:16**	**15:0**
MFX Common	011	10	000	000	subopcode	DWord Count
Reserved	011	10	000	001-111	subopcode	DWord Count
AVC Common	011	10	001	000	subopcode	DWord Count
AVC Dec	011	10	001	001	subopcode	DWord Count
AVC Enc	011	10	001	010	subopcode	DWord Count
Reserved	011	10	001	011-111	subopcode	DWord Count
Reserved (for VC1 Common)	011	10	010	000	subopcode	DWord Count
VC1 Dec	011	10	010	001	subopcode	DWord Count

Bits							
TYPE	31:29	28:24		23	22	21:0	
Reserved (for VC1 Enc)	011	10	010	010		subopcode	DWord Count
Reserved	011	10	010	011-111		subopcode	DWord Count
Reserved (MPEG2 Common)	011	10	011	000		subopcode	DWord Count
MPEG2 Dec	011	10	011	001		subopcode	DWord Count
Reserved (for MPEG2 Enc)	011	10	011	010		subopcode	DWord Count
Reserved	011	10	011	011-111		subopcode	DWord Count
Reserved	011	10	100-111	XXX			

5.2 Command Map

This section provides a map of the graphics command opcodes.

5.2.1 Memory Interface Command Map

All the following commands are defined in *Memory Interface Commands*.

Table 5-3. Memory Interface Commands for RCP

Opcode (28:23)	Command	Pipe		
		Render	Video	Blitter [DevSNB+]
1-DWord				
00h	MI_NOOP	All	All	All
01h	Reserved			
02h	MI_USER_INTERRUPT	All	All	All
03h	MI_WAIT_FOR_EVENT	All	All	All
04h	MI_FLUSH	All	[pre-DevGT]	
05h	MI_ARB_CHECK	All	All	All
06h	Reserved			
07h	MI_REPORT_HEAD	All	All	All
08h	MI_ARB_ON_OFF	All	[DevSNB+]	
09h	Reserved			
0Ah	MI_BATCH_BUFFER_END	All	All	All

Opcode (28:23)	Command	Pipe		
		Render	Video	Blitter [DevSNB+]
0Bh	MI_SUSPEND_FLUSH	[DevSNB+]		
0Ch	Reserved			
0Dh	Reserved			
0Eh	Reserved			
0Fh	Reserved			
2+ DWord				
10h	Reserved			
11h	MI_OVERLAY_FLIP Reserved [DevCTG+]	[pre-DevCTG]		
12h	MI_LOAD_SCAN_LINES_INCL Reserved [DevSNB+]	[pre-DevSNB]		
13h	MI_LOAD_SCAN_LINES_EXCL Reserved [DevSNB+]	[pre-DevSNB]		
14h	MI_DISPLAY_FLIP	All		
15h	Reserved			
16h	MI_SEMAPHORE_MBOX	[DevCTG+]	[DevSNB+]	[DevSNB+]
17h	Reserved			
18h	MI_SET_CONTEXT	All		
19h	Reserved			
1Ah	MI_MATH			
1Bh-1Fh	Reserved			
Store Data				
20h	MI_STORE_DATA_IMM	All	All	All
21h	MI_STORE_DATA_INDEX	All	All	All
22h	MI_LOAD_REGISTER_IMM	All	All	All
23h	MI_UPDATE_GTT	[DevCTG+]	[DevSNB+]	
24h	MI_STORE_REGISTER_MEM	All	All	All
25h	MI_PROBE	[DevCTG] [DevILK]		
26h	MI_FLUSH_DW [DevILK] This is the opcode for MI_REPORT_PERF_COUNT. It only applied to Render pipe		[DevSNB+]	[DevSNB+]
27h	MI_CLFLUSH	[DevSNB+]		
28h	MI_REPORT_PERF_COUNT	[DevSNB+]		
29h	Reserved			
2Ah	Reserved			

Opcode (28:23)	Command	Pipe		
		Render	Video	Blitter [DevSNB+]
2Bh	Reserved			
2Ch–2Fh	Reserved			
Ring/Batch Buffer				
30h	Reserved			
31h	MI_BATCH_BUFFER_START	All	All	All
32h–35h	Reserved			
36h	MI_CONDITIONAL_BATCH_BUFFER_END	[DevSNB+]	[DevSNB+]	
37h–3Fh	Reserved			

2D Command Map

All the following commands are defined in *Blitter Instructions*.

Opcode (28:22)	Command
00h	Reserved
01h	XY_SETUP_BLT
02h	Reserved
03h	XY_SETUP_CLIP_BLT
04h–10h	Reserved
11h	XY_SETUP_MONO_PATTERN_SL_BLT
12h–23h	Reserved
24h	XY_PIXEL_BLT
25h	XY_SCANLINES_BLT
26h	XY_TEXT_BLT
23h–30h	Reserved
31h	XY_TEXT_IMMEDIATE_BLT
32h–3Fh	Reserved
40h	COLOR_BLT
41h–42h	Reserved
43h	SRC_COPY_BLT
44h–4Fh	Reserved
50h	XY_COLOR_BLT
51h	XY_PAT_BLT
52h	XY_MONO_PAT_BLT
53h	XY_SRC_COPY_BLT
54h	XY_MONO_SRC_COPY_BLT
55h	XY_FULL_BLT

Opcode (28:22)	Command
56h	XY_FULL_MONO_SRC_BLT
57h	XY_FULL_MONO_PATTERN_BLT
58h	XY_FULL_MONO_PATTERN_MONO_SRC_BLT
59h	XY_MONO_PAT_FIXED_BLT
5Ah–70h	Reserved
71h	XY_MONO_SRC_COPY_IMMEDIATE_BLT
72h	XY_PAT_BLT_IMMEDIATE
73h	XY_SRC_COPY_CHROMA_BLT
74h	XY_FULL_IMMEDIATE_PATTERN_BLT
75h	XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT
76h	XY_PAT_CHROMA_BLT
77h	XY_PAT_CHROMA_BLT_IMMEDIATE
78h–7Fh	Reserved

3D/Media Command Map

Pipeline Type (28:27)	Opcode	Sub Opcode	Command	Definition Chapter
3D State (Pipelined)	**Bits 26:24**	**Bits 23:16**		
3h	0h	00h	3DSTATE_PIPELINED_POINTERS [Pre-DevSNB]	3D Pipeline
3h	0h	01h	3DSTATE_BINDING_TABLE_POINTERS [DevSNB]	3D Pipeline
3h	0h	02h	3DSTATE_SAMPLER_STATE_POINTERS [DevSNB]	3D Pipeline
3h	0h	03h	Reserved	n/a
3h	0h	04h	Reserved [DevSNB]	
3h	0h	05h	3DSTATE_URB [DevSNB]	3D Pipeline
3h	0h	06h	Reserved [DevSNB]	n/a
3h	0h	07h	Reserved [DevSNB]	n/a
3h	0h	08h	3DSTATE_VERTEX_BUFFERS	Vertex Fetch
3h	0h	09h	3DSTATE_VERTEX_ELEMENTS	Vertex Fetch
3h	0h	0Ah	3DSTATE_INDEX_BUFFER	Vertex Fetch
3h	0h	0Bh	3DSTATE_VF_STATISTICS	Vertex Fetch
3h	0h	0Ch	Reserved	n/a
3h	0h	0Dh	3DSTATE_VIEWPORT_STATE_POINTERS [DevSNB]	3D Pipeline

Pipeline Type (28:27)	Opcode	Sub Opcode	Command	Definition Chapter
3h	0h	0Eh	3DSTATE_CC_STATE_POINTERS [DevSNB]	3D Pipeline
3h	0h	0Fh	3DSTATE_SCISSOR_STATE_POINTERS [DevSNB]	3D Pipeline
--	--	--	Reserved [DevSNB]	
3h	0h	11h	3DSTATE_GS [DevSNB+]	Geometry Shader
3h	0h	12h	3DSTATE_CLIP [DevSNB+]	Clipper
--	--	--	Reserved [DevSNB+]	
3h	0h	14h	3DSTATE_WM [DevSNB+]	Windower
3h	--	--	Reserved [DevSNB+]	
3h	0h	16h	3DSTATE_CONSTANT_GS [DevSNB+]	Geometry Shader
3h	0h	17h	3DSTATE_CONSTANT_PS [DevSNB+]	Windower
3h	0h	18h	3DSTATE_SAMPLE_MASK [DevSNB+]	Windower
3D State (Non-Pipelined)	**Bits 26:24**	**Bits 23:16**		
3h	1h	00h	3DSTATE_DRAWING_RECTANGLE	Strips & Fans
3h	1h	01h	3DSTATE_CONSTANT_COLOR [Pre-DevSNB]	Color Calculator
3h	1h	02h	3DSTATE_SAMPLER_PALETTE_LOAD0	Sampling Engine
3h	1h	03h	Reserved	
3h	1h	04h	3DSTATE_CHROMA_KEY	Sampling Engine
3h	1h	05h	3DSTATE_DEPTH_BUFFER [DevSNB]	Windower
3h	1h	06h	3DSTATE_POLY_STIPPLE_OFFSET	Windower
3h	1h	07h	3DSTATE_POLY_STIPPLE_PATTERN	Windower
3h	1h	08h	3DSTATE_LINE_STIPPLE	Windower
3h	1h	09h	3DSTATE_GLOBAL_DEPTH_OFFSET_CLAMP [Pre-DevSNB]	Windower
3h	1h	0Ah	[DevCTG]: 3DSTATE_AA_LINE_PARAMS [DevCTG+]	Windower
3h	1h	0Bh	3DSTATE_GS_SVB_INDEX [DevCTG+]	Geometry Shader
3h	1h	0Ch	3DSTATE_SAMPLER_PALETTE_LOAD1 [DevCTG-B+]	Sampling Engine

Pipeline Type (28:27)	Opcode	Sub Opcode	Command	Definition Chapter
3h	1h	0Dh	3DSTATE_MULTISAMPLE [DevSNB+]	Windower
3h	1h	0Eh	3DSTATE_STENCIL_BUFFER [DevIL,DevSNB] Reserved [Pre-DevILK]	Windower
3h	1h	0Fh	3DSTATE_HIER_DEPTH_BUFFER [DevILK, DevSNB] Reserved [Pre-DevILK]	Windower
3h	1h	10h	3DSTATE_CLEAR_PARAMS [DevIL, DevSNB]	Windower
3h	1h	11h	3DSTATE_MONOFILTER_SIZE [DevILK+]	Sampling Engine
3h	1h	12h	Reserved	--
3h	1h	13h	Reserved	--
3h	1h	14h	Reserved	--
3h	1h	15h	Reserved	--
3h	1h	16h	Reserved	--
3h	1h	17h	3DSTATE_SO_DECL_LIST	HW Streamout
3h	1h	18h	3DSTATE_SO_BUFFER	HW Streamout
3h	1h	19h–FFh	Reserved	n/a
3D (Control)	**Bits 26:24**	**Bits 23:16**		
3h	2h	00h	PIPE_CONTROL	3D Pipeline
3h	2h	01h–FFh	Reserved	n/a
3D (Primitive)	**Bits 26:24**	**Bits 23:16**		
3h	3h	00h	3DPRIMITIVE	Vertex Fetch
3h	3h	01h–FFh	Reserved	n/a
3h	4h–7h	00h–FFh	Reserved	n/a

5.2.2 Video Codec Command Map

5.2.2.1 MFX Common Command Map [DevSNB+]

MFX state commands support direct state model and indirect state model. Recommended usage of indirect state model is provided here (as a software usage guideline).

Pipeline Type (28:27)	Opcode (26:24)	Subop A (23:21)	Subop B (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
MFX Common (State)							
2h	0h	0h	0h	MFX_PIPE_MODE_SELECT	MFX	IMAGE	n/a
2h	0h	0h	1h	MFX_SURFACE_STATE	MFX	IMAGE	n/a
2h	0h	0h	2h	MFX_PIPE_BUF_ADDR_STATE	MFX	IMAGE	n/a
2h	0h	0h	3h	MFX_IND_OBJ_BASE_ADDR_STATE	MFX	IMAGE	n/a
2h	0h	0h	4h	MFX_BSP_BUF_BASE_ADDR_STATE	MFX	IMAGE	n/a
2h	0h	0h	5h	Reserved	n/a	n/a	n/a
2h	0h	0h	6h	MFX_ STATE_POINTER	MFX	IMAGE	n/a
2h	0h	0h	7-8h	Reserved	n/a	n/a	n/a
MFX Common (Object)							
2h	0h	1h	9h	MFD_ IT_OBJECT	MFX	n/a	**Yes**
2h	0h	0h	4-1Fh	Reserved	n/a	n/a	n/a
AVC Common (State)							
2h	1h	0h	0h	MFX_AVC_IMG_STATE	MFX	IMAGE	n/a
2h	1h	0h	1h	MFX_AVC_QM_STATE	MFX	IMAGE	n/a
2h	1h	0h	2h	MFX_AVC_DIRECTMODE_STATE	MFX	SLICE	n/a
2h	1h	0h	3h	MFX_AVC_SLICE_STATE	MFX	SLICE	n/a
2h	1h	0h	4h	MFX_AVC_REF_IDX_STATE	MFX	SLICE	n/a
2h	1h	0h	5h	MFX_AVC_WEIGHTOFFSET_STATE	MFX	SLICE	n/a
2h	1h	0h	6-1Fh	Reserved	n/a	n/a	n/a
AVC Dec							
2h	1h	1h	0-7h	Reserved	n/a	n/a	n/a
2h	1h	1h	8h	MFD_AVC_BSD_OBJECT	MFX	n/a	**No**

Pipeline Type (28:27)	Opcode (26:24)	Subop A (23:21)	Subop B (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
2h	1h	1h	9-1Fh	Reserved	n/a	n/a	n/a
AVC Enc							
2h	1h	2h	0-1h	Reserved	n/a	n/a	n/a
2h	1h	2h	2h	MFC_AVC_FQM_STATE	MFX	IMAGE	n/a
2h	1h	2h	3-7h	Reserved	n/a	n/a	n/a
2h	1h	2h	8h	MFC_AVC_PAK_INSERT_OBJECT	MFX	n/a	n/a
2h	1h	2h	9h	MFC_AVC_PAK_OBJECT	MFX	n/a	**Yes**
2h	1h	2h	A-1Fh	Reserved	n/a	n/a	n/a
2h	1h	2h	0-1Fh	Reserved	n/a	n/a	n/a
VC1 Common							
2h	2h	0h	0h	MFX_VC1_PIC_STATE	MFX	IMAGE	n/a
2h	2h	0h	1h	MFX_VC1_PRED_PIPE_STATE	MFX	IMAGE	n/a
2h	2h	0h	2h	MFX_VC1_DIRECTMODE_STATE	MFX	SLICE	n/a
2h	2h	0h	2-1Fh	Reserved	n/a	n/a	n/a
VC1 Dec							
2h	2h	1h	0-7h	Reserved	n/a	n/a	n/a
2h	2h	1h	8h	MFD_VC1_BSD_OBJECT	MFX	n/a	**Yes**
2h	2h	1h	9-1Fh	Reserved	n/a	n/a	n/a
VC1 Enc							
2h	2h	2h	0-1Fh	Reserved	n/a	n/a	n/a
MPEG2 Common							
2h	3h	0h	0h	MFX_MPEG2_PIC_STATE	MFX	IMAGE	n/a
2h	3h	0h	1h	MFX_MPEG2_QM_STATE	MFX	IMAGE	n/a
2h	3h	0h	2-1Fh	Reserved	n/a	n/a	n/a
MPEG2 Dec							

Pipeline Type (28:27)	Opcode (26:24)	Subop A (23:21)	Subop B (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interrupt table?
2h	3h	1h	1-7h	Reserved	n/a	n/a	n/a
2h	3h	1h	8h	MFD_MPEG2_BSD_OBJECT	MFX	n/a	**Yes**
2h	3h	1h	9-1Fh	Reserved	n/a	n/a	n/a
MPEG2 Enc							
2h	3h	2h	0-1Fh	Reserved	n/a	n/a	n/a
The Rest							
2h	4-5h, 7h	x	x	Reserved	n/a	n/a	n/a

6. Register Address Maps

6.1 Graphics Register Address Map

This chapter provides address maps of the graphics controllers I/O and memory-mapped registers. Individual register bit field descriptions are provided in the following chapters. PCI configuration address maps and register bit descriptions are provided in the following chapter.

6.1.1 Memory and I/O Space Registers [DevSNB+]

These are graphics MMIO ranges used for DevSNB. Note that this is only a subset of the complete definition of the MMIO address space.

Range Start	Range End	Unit owner
00002000	00002FFF	Render/Generic Media Engine
00004000	00004FFF	Render/Generic Media Graphics Memory Arbiter
--	--	Reserved
--	--	Reserved
00012000	000123FF	MFX Control Engine (Video Command Streamer)
00012400	00012FFF	Media Units (VIN unit)
00014000	00014FFF	MFX Memory Arbiter
00022000	00022FFF	Blitter Engine
00024000	00024FFF	Blitter Memory Arbiter
--	--	Reserved
00100000	00107FFF	Fence Registers
00140000	0017FFFF	MCHBAR (SA)

Note: 8800-88FF is a reserved range for DevSNB. IA accesses to this region has no impact however TAP (backdoor) accesses to reserved range will result in hardware hang. Do not use.

6.1.2 Graphics Register Memory Address Map [DevSNB]

All graphics device registers are directly accessible via memory-mapped I/O and indirectly accessible via the MMIO_INDEX and MMIO_DATA I/O registers. In addition, the VGA and Extended VGA registers are I/O mapped.

Table 6-1. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
00000h–00FFFh	—	**VGA and VGA Extended Registers** These registers are both memory and I/O mapped and are listed in the following table. Note that the I/O address and memory offset address are the same value for each register.	—
Reserved (1000h–1FFFh)			
01000h–01FFFh	—	Reserved	—
Primary CS Instruction and Interrupt Control Registers (02000h–02FFFh)			
02000h–0201Fh	—	Reserved	—
02020h–02023h	PGTBL_CTL	Page Table Control Register	R/W
02024h–02027h	--	Reserved	--
02028h–0202Bh	EXCC	Execute Condition Code Register	R/W,RO
0202Ch–0202Fh	—	Reserved	—
02030h–02033h	PRB0_TAIL	Primary Ring Buffer 0 Tail Register	R/W
02034h–02037h	PRB0_HEAD	Primary Ring Buffer 0 Head Register	R/W
02038h–0203Bh	PRB0_STARTsted	Primary Ring Buffer 0 Start Register	R/W
0203Ch–0203Fh	PRB0_CTL	Primary Ring Buffer 0 Control Register	R/W
02040h–0205Fh	—	Reserved	—
02060h–02063h	--	Reserved	--
02064h–02067h	--	Reserved	--
02068h–0206Bh	--	Reserved	--
0206Ch–0206Fh	--	Reserved	--
02070h–02073h	--	Reserved	--
02074h–02077h	--	Reserved	--
02078h–0207Bh	--	Reserved	--
0207Ch–0207Fh	--	Reserved	--
02080h–02083h	HWS_PGA	Hardware Status Page Address Register	R/W
02084h–02087h	—	Reserved	—

Table 6-1. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
02088h–0208Ch	--		R/W
0208Dh–02093h	—	Reserved	—
02094h–02097h	NOPID	NOP Identification Register	RO
02098h–0209Bh	HWSTAM	Hardware Status Mask Register	R/W
0209Ch–0209Fh	MI_MODE	Mode Register for Software Interface	R/W
020A0h–020A3h	IER	Interrupt Enable Register	R/W
020A4h–020A7h	IIR	Interrupt Identity Register	R/WC
020A8h–020ABh	IMR	Interrupt Mask Register	R/W
020ACh–020AFh	ISR	Interrupt Status Register	RO
020B0h–020B3h	EIR	Error Identity Register	R/WC
020B4h–020B7h	EMR	Error Mask Register	R/W
020B8h–020BBh	ESR	Error Status Register	RO
020BCh–020BFh	—	Reserved	—
020C0h–020C3h	INSTPM	Instruction Parser Mode Register (SAVED/RESTORED)	R/W
020C4h–020C7h	--		R/W
020C8h–020CBh	--		R/W
020CCh–020DFh	—	Reserved	—
020E0h–020E3h	--		R/W
	--		R/W
020E4h–020E7h	MI_ARB_STATE	Memory Interface Arbitration State Register (SAVED/RESTORED)	R/W
020E8h–020FBh	—	Reserved	—
020FCh–020FFh	--		R/W
02100h–0210Fh	—	Reserved	—
02110h–02113h	BB_STATE	Batch Buffer State Register	R/W
02114h–0211Fh	—	Reserved	—
02120h–02123h	--	Reserved	--
02124h–02127h	--	Reserved	--
02128h–02133h	—	Reserved	—
02134h–02137h	UHPTR	Pending Head Pointer Register	R/W
02138h–0213Fh	—	Reserved	—

Table 6-1. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
02140h–02147h	BB_ADDR	Batch Buffer Current Address	RO
0214Ch–0216Fh	—	Reserved	—
02170h–02177h	GFX_FLSH_CNTL	Graphics Flush Control	R/W
02178h–0217Bh	PR_CTR_CTL	Render Watchdog Counter Control	R/W
0217Ch–0217Fh	PR_CTR_THRSH	Render Watchdog Counter Threshold	R/W
02180h–02183h	CCID0	Current Context ID 0 (assoc w/ PRB0)	R/W
02184h–0218Fh	—	Reserved	—
02190h–02193h	PR_CTR	Render Watchdog Counter	RO
02194h–0219Fh	—	Reserved	—
021A0h–021A3h	--	Reserved	--
021A4h–021A7h	--	Reserved	--
021A8h-021CFh	—	Reserved	—
021D0h-021D3h	--	Reserved	--
021D4h-021FFh	—	Reserved	—
02200h–02303h	--	Reserved	--
02204h–02207h	--	Reserved	--
02208h–0220Bh	--	Reserved	--
0220Ch–0220Fh	--	Reserved	--
02210h–02213h	--	Reserved	--
02214h–0230Fh	—	Reserved	—
02310h-02317h	IA_VERTICES_COUNT	Reported Vertices Counter	R/W
02318h-0231Fh	IA_PRIMITVES_COUNT	Reported Vertex Fetch Output Primitives Counter	R/W
02320h-02327h	--	Reserved	--
02328h-0232Fh	GS_INVOCATION_COUNT	Reported Geometry Shader Thread Invocation Counter	R/W
02330h-02337h	GS_PRIMITIVES_COUNT	Reported Geometry Shader Output Primitives Counter	R/W
02338h-0233Fh	CL_INVOCATION_COUNT	Reported Clipper Thread Invocation Counter	R/W
02340h-02347h	CL_PRIMITIVES_COUNT	Reported Clipper Output Primitives Counter	R/W

Table 6-1. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
02348h-0234Fh	PS_INVOCATION_COUNT	Reported Fragments Shaded Invocation Counter	R/W
02350h-02357h	PS_DEPTH_COUNT	Reported Fragments Passing Depth Test Counter	R/W
02358–0235Fh	TIMESTAMP	Reported Timestamp Count	R/W
02360–02367h	--	Reserved	--
02368h–0236Fh	—	Reserved	—
02370h–02377h	--		R/W
02378h–0237Fh	--		R/W
02380h–02387h			R/W
02388h–0244Fh	—	Reserved	—
02450h–02453h	--	Reserved	--
02454h–0246Fh	—	Reserved	—
02470h–02473h	--	Reserved	--
02474h–024FFh	—	Reserved	—
FENCE & Per-Process GTT Control (03000h–031FFh)			
03000h-03007h	FENCE[0]	Graphics Memory Fence Table Register [0]	R/W
…	…	…	
0307Ch-0307Fh	FENCE[15]	Graphics Memory Fence Table Register [15]	R/W
BCS Instruction and Interrupt Control Registers (04000h–043FFh)			
04000h–04023h	—	Reserved	—
04024h–04027h	--	Reserved	--
04028h–0402Fh	—	Reserved	—
04030h–04033h	BCS_RB_TAIL	Ring Buffer Tail Register	R/W
04034h–04037h	BCS_RB_HEAD	Ring Buffer Head Register	R/W
04038h–0403Bh	BCS_RB_START	Ring Buffer Start Register	R/W
0403Ch–0403Fh	BCS_RB_CTL	Ring Buffer Control Register	R/W
04040h–04063h	—	Reserved	—
04064h–04067h	--	Reserved	--
04068h–0406Bh	—	Reserved	—
0406Ch–04073h	—	Reserved	—
04074h–04077h	—	Reserved	—
04078h–0407Bh	—	Reserved	—
0407Ch–0407F	—	Reserved	—

Table 6-1. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
04080h–04083h	BCS_HWS_PGA	Hardware Status Page Address	R/W
04084h–04079	—	Reserved	—
04094h–04097h	BCS_NOPID	NOP Identification Register	RO
04098h–0409Bh	—	Reserved	—
0409Ch–0409Fh	BCS_MI_MODE	Mode Register for Software Interface	R/W
040A0h–040BFh	—	Reserved	—
040C0h–040C3h	BCS_INSTPM	Instruction Parser Mode Register	R/W
040C4h–04133h	—	Reserved	—
04134h–04137h	BCS_UHPTR	Pending Head Pointer Register	R/W
04138h–0413Fh	—	Reserved	—
04140h–04143h	BCS_BB_STR	Batch Buffer Start Register	RO
04144h–0418Fh	—	Reserved	—
04190h–04193h	BCS_RCCID	Current Context ID	RO
04194h–04197h	BCS_RNCID	Next Context ID	R/W
04198h–041CFh	—	Reserved	—
041D0h-041D3h	—	Reserved	—
041D4h–0438Fh	—	Reserved	—
04390h–04393h	—	Reserved	—
04394h–04397h	—	Reserved	—
04398h–0439Fh	—	Reserved	—
043A0h–043FFh	—	Reserved	—
Reserved Registers (04400h–044FFh)			
MFC Status Registers (012400h–012444h) **For [DevSNB+] only; Otherwise, this Range is Reserved.**			
	—	Reserved	—
12400h	AVD Error flags	AVD Internal Error flags	RW
12404h	AVD Error counter	AVD Error Counter	RW
12408h	MFC_BITSTREAM_BYTE COUNT_SLICE	Bitstream Output Byte Count Register per Slice	RO
1240Ch	MFC_BITSTREAM_SE_BI TCOUNT_SLICE	Bitstream Output Bit Count for the last Syntax Element Register	RO
12410h	MFC_AVC_CABAC_INSE RTION_COUNT	Bitstream Output CABAC Insertion Count Register	RO

Table 6-1. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
12414h	MFC_AVC_MINSIZE_PADDING_COUNT	Bitstream Output Minimal Size Padding Count Register	RO
12418h	MFC_IMAGE_STATUS_MASK	Image Coding Status Mask Register	RO
1241Ch	MFC_IMAGE_STATUS_CONTROL	Image Coding Status Control Register	RO
12420h	MFC_BITSTREAM_BYTECOUNT_FRAME	Total Bitstream Output Byte Count register per Frame	RO
12424h	MFC_BITSTREAM_SE_BITCOUNT_FRAME	Bitstream Output total Byte Count for syntax eements (total byes of MB data from SEC per frame)	RO
12428h	MFC_AVC_CABAC_BIN_COUNT_FRAME	Bitstream Output total bin count per frame	RO
1242Ch	—	Reserved	—
12430h	—	Reserved	—
12434h	—	Reserved	—
12438h	—	Reserved	—
1243Ch	—	Reserved	—
12440h	—	Reserved	—
12444h	—	Reserved	—
	—	Reserved	—
VSC Registers (05000h – 05FFFh)			
05000h-05003h		FMD Variance 0 for Stream 0	
05004h-05007h		FMD Variance 1 for Stream 0	
05008h-0500Bh		FMD Variance 2 for Stream 0	
0500Ch-0500Fh		FMD Variance 3 for Stream 0	
05010h-05013h		FMD Variance 4 for Stream 0	
05014h-05017h		FMD Variance 5 for Stream 0	
05018h-0501Bh		FMD Variance 6 for Stream 0	
0501Ch-0501Fh		FMD Variance 7 for Stream 0	
05020h-05023h		FMD Variance 8 for Stream 0	
05024h-05027h		FMD Variance 9 for Stream 0	
05028h-0502Bh		FMD Variance 10 for Stream 0	

Table 6-1. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
0502Ch-0502Fh		GNE Sum for Stream0	
05030h-05033h		Number of Valid GNE Blocks Strm0	
05034h-05037h		FMD Variance 0 for Stream 1	
05038h-0503Bh		FMD Variance 1 for Stream 1	
0503Ch-0503Fh		FMD Variance 2 for Stream 1	
05040h-05043h		FMD Variance 3 for Stream 1	
05044h-05047h		FMD Variance 4 for Stream 1	
05048h-0504Bh		FMD Variance 5 for Stream 1	
0504Ch-0504Fh		FMD Variance 6 for Stream 1	
05050h-05053h		FMD Variance 7 for Stream 1	
05054h-05057h		FMD Variance 8 for Stream 1	
05058h-0505Bh		FMD Variance 9 for Stream 1	
0505Ch-0505Fh		FMD Variance 10 for Stream 1	
05060h-05063h		GNE Sum for Stream 1	
05064h-05067h		Number of Valid GNE Blocks Strm1	
05068h-0506Fh	—	Reserved	
VSC Registers **For [DevSNB+] only, otherwise reserved**			
05070h-05073h		Ymax (bits 25:16]), Ymin (bits[9:0]), other bits zero	
05074h-05077h		Number of skin pixels (bits [20:0], other bits zero)	
05078h-0507Fh	—	Reserved	
05080h-05081h		ACE Histogram, bin 0	
05082h-05083h		ACE Histogram, bin 1	
…		ACE Histogram, bins 2 - 126	
0517Eh-0517Fh		ACE Histogram, bin 127	
Clock Control and Power Management Registers (06000h–06FFFh)			
06000h–06003h	VGA0	VGA 0 Divisor	R/W
06004h–06007h	VGA1	VGA 1 Divisor	R/W
06008h–0600Fh		Reserved	
06010h–06013h	VGA_PD	VGA Post Divisor Select	R/W
06014h–06017h	DPLLA_CTRL	Display PLL A Control	R/W
06018h–0601Bh	DPLLB_CTRL	Display PLL B Control	R/W
0601Ch–0601Fh	--	Reserved	--

Table 6-1. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
06020h–06023h	--	Reserved	--
06024h–0603Fh	—	Reserved	—
06040h–06043h	FPA0	DPLL A Divisor 0	R/W
06044h–06047h	FPA1	DPLL A Divisor 1	R/W
06048h–0604Bh	FPB0	DPLL B Divisor 0	R/W
0604Ch–0604Fh	FPB1	DPLL B Divisor 1	R/W
06050h–0606Bh	—	Reserved	—
0606Ch–0606Fh	--	Reserved	--
06070h–06103h	—	Reserved	—
06104h–06107h	D_STATE	D State Function Control	R/W
06108h–061FFh	—	Reserved	—
06200h–06203h	DSPCLK_GATE_D	Clock Gating Disable for Display Register	R/W
06204h–06207h	RENCLK_GATE_D1	Clock Gating Disable for Render Register I	R/W
06208h–0620Bh	RENDCLK_GATE_D2	Clock Gating Disable for Render Register II	—
0620Ch–0620Fh	VDECCLK_GATE_D	Clock Gating Disable for Video Decode Register ([DevCTG] Only)	R/W
06210h–06213h	--		R/W
06214h–06125h	--	Reserved	--
06216h–06FFFh	—	Reserved	—
Reserved Registers (07000h–073FFh)			
Reserved Registers (07400h–088FFh)			
Reserved Registers (08900h–09FFFh)			
Display Palette (0A000h–0AFFFh)			
0A000h–0A3FFh	DPALETTE_A	Display Pipe A Palette	R/W
0A400h–0A7FFh	—	Reserved	—
0A800h–0ABFFh	DPALETTE_B	Display Pipe B Palette	R/W
0AC00h–0AFFFh	—	Reserved	—
GFX MMIO – MCHBAR Aperture (10000h-13FFFh)			
10000h-13FFFh	—	MCHBAR Aperture	R/W
Reserved (14000h–2FFFFh)			
14000h-2FFFFh	—	Reserved	—

Table 6-1. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
Overlay Registers (30000h–03FFFFh) (For additional address offsets in the double-buffering scheme, see Overlay Chapter)			
30000h–30003h	OVADD	Overlay Register Update Address	R/W
30004h–30007h	--	Reserved	--
30008h–3000Bh	DOVSTA	Display/Overlay Status	RO
3000Ch–3000Fh	DOVSTAEX	Display/Overlay Extended Status	RO
30010h–30013h	OVR_GAMMA5	Overlay Gamma Correction [5]	R/W
30014h–30017h	OVR_GAMMA4	Overlay Gamma Correction [4]	R/W
30018h–3001Bh	OVR_GAMMA3	Overlay Gamma Correction [3]	R/W
3001Ch–3001Fh	OVR_GAMMA2	Overlay Gamma Correction [2]	R/W
30020h–30023h	OVR_GAMMA1	Overlay Gamma Correction [1]	R/W
30024h–30027h	OVR_GAMMA0	Overlay Gamma Correction [0]	R/W
30028h–30057h	—	Reserved	—
30058h–3005Bh	SYNCPH0	Overlay Flip Sync Lock Phase 0	RO
3005Ch–3005Fh	SYNCPH1	Overlay Flip Sync Lock Phase 1	RO
30060h–30063h	SYNCPH2	Overlay Flip Sync Lock Phase 2	RO
30064h–30067h	SYNCPH3	Overlay Flip Sync Lock Phase 3	RO
30068h–300FFh	—	Reserved	—
30100h–30103	OBUF_0Y	Overlay Buffer 0 Y Pointer	RO
30104h–30107h	OBUF_1Y	Overlay Buffer 1 Y Pointer	RO
30108h–3010Bh	OBUF_0U	Overlay Buffer 0 U Pointer	RO
3010Ch–3010Fh	OBUF_0V	Overlay Buffer 0 V Pointer	RO
30110h–30113h	OBUF_1U	Overlay Buffer 1 U Pointer	RO
30114h–30117h	OBUF_1V	Overlay Buffer 1 V Pointer	RO
30118h–3011Bh	OSTRIDE	Overlay Stride	RO
3011Ch–3011Fh	YRGB_VPH	Y/RGB Vertical Phase	RO
30120h–30123h	UV_VPH	UV Vertical Phase	RO
30124h–30127h	HORZ_PH	Horizontal Phase	RO
30128h–3012Bh	INIT_PHS	Initial Phase	RO
3012Ch–3012Fh	DWINPOS	Destination Window Position	RO
30130h–30133h	DWINSZ	Destination Window Size	RO
30134h–30137h	SWIDTH	Source Width	RO
30138h–3013Bh	SWIDTHSW	Source Width in Swords	RO
3013Ch–3013Fh	SHEIGHT	Source Height	RO

Table 6-1. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
30140h–30143h	YRGBSCALE	Y/RGB Scale Factor	RO
30144h–30147h	UVSCALE	U V Scale Factor	RO
30148h–3014Bh	OVCLRC0	Overlay Color Correction 0	RO
3014Ch–3014Fh	OVCLRC1	Overlay Color Correction 1	RO
30150h–30153h	DCLRKV	Destination Color Key Value	RO
30154h–30157h	DCLRKM	Destination Color Key Mask	RO
30158h–3015Bh	SCHRKVH	Source Chroma Key Value High	RO
3015Ch–3015Fh	SCHRKVL	Source Chroma Key Value Low	RO
30160h–30163h	SCHRKEN	Source Chroma Key Enable	RO
30164h–30167h	OCONFIG	Overlay Configuration	RO
30168h–3016Bh	OCMD	Overlay Command	RO
3016Ch–3016Fh	—	Reserved	—
30170h–30173h	OSTART_0Y	Overlay Surface Y 0 Base Address Register	RO
30174h–30177h	OSTART _1Y	Overlay Surface Y 1 Base Address Register	RO
30178h–3017Bh	OSTART _0U	Overlay Surface U 0 Base Address Register	RO
3017Ch–3017Fh	OSTART _0V	Overlay Surface V 0 Base Address Register	RO
30180h–30183h	OSTART _1U	Overlay Surface U 1 Base Address Register	RO
30184h–30187h	OSTART _1V	Overlay Surface V 1 Base Address Register	RO
30188h–3018Bh	OTILEOFF_0Y	Overlay Surface Y 0 Base Address Register	RO
3018Ch–3018Fh	OTILEOFF _1Y	Overlay Surface Y 1 Base Address Register	RO
30190h–30193h	OTILEOFF _0U	Overlay Surface U 0 Bae Address Register	RO
30194h–30197h	OTILEOFF _0V	Overlay Surface V 0 Base Address Register	RO
30198h–3019Bh	OTILEOFF _1U	Overlay Surface U 1 Base Address Register	RO
3019Ch–3019Fh	OTILEOFF _1V	Overlay Surface V 1 Base Address Register	RO
301A0h–301A3h	—	Reserved	—
301A4h–301A7h	UVSCALEV	UV Vertical Downscale Integer Register	RO
301A8h–302FFh	—	Reserved	—
30300h–303FFh	Y_VCOEFS	Overlay Y Vertical Filter Coefficients	RO
30368h–303FFh	—	Reserved	—
30400h–305FFh	Y_HCOEFS	Overlay Y Horizontal Filter Coefficient	RO
304ACh–305FFh	—	Reserved	—
30600h–306FFh	UV_VCOEFS	Overlay UV Vertical Filter Coefficients	RO
30668h–306FFh	—	Reserved	—

Table 6-1. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
30700h–307FFh	UV_HCOEFS	Overlay UV Horizontal Filter Coefficients	RO
30768h–3FFFFh	—	Reserved	—
Reserved (40000h–5FFFFh)			
40000h–5FFFFh	—	Reserved	—
Display Engine Pipeline Registers (60000h–6FFFFh)			
Display Pipeline A			
60000h–60003h	HTOTAL_A	Pipe A Horizontal Total	R/W
60004h–60007h	HBLANK_A	Pipe A Horizontal Blank	R/W
60008h–6000Bh	HSYNC_A	Pipe A Horizontal Sync	R/W
6000Ch–6000Fh	VTOTAL_A	Pipe A Vertical Total	R/W
60010h–60013h	VBLANK_A	Pipe A Vertical Blank	R/W
60014h–60017h	VSYNC_A	Pipe A Vertical Sync	R/W
60018h–6001Bh	—	Reserved	R/W
6001Ch–6001Fh	PIPEASRC	Pipe A Source Image Size	R/W
60020h–60023h	--	Reserved	--
60024h–60027h	—	Reserved	—
60028h–6002Bh	VSYNCSHIFT_A	Vertical Sync Shift Register A	—
6002Ch–6004Fh	—	Reserved	—
60050h–60053h	CRCCTRLREDA	Pipe A CRC Red Control	R/W
60054h–60057h	CRCCTRLGREENA	Pipe A CRC Green Control	R/W
60058h–6005Bh	CRCCTRLBLUEA	Pipe A CRC Blue Control	R/W
6005Ch–6005Fh	CRCCTRLRESA	Pipe A CRC Residual Control Register	R/W
60060h–60063h	CRCRESREDA	Pipe A CRC Red Result	RO
60064h–60067h	CRCRESGREENA	Pipe A CRC Green Result	RO
60068h–6006Bh	CRCRESBLUEA	Pipe A CRC Blue Result	RO
6006Ch-6006Fh	CRCRESRESA	Pipe A CRC Residual Result	RO
60070h–60FFFh	—	Reserved	—
Display Pipeline B			
61000h–61003h	HTOTAL_B	Pipe B Horizontal Total	R/W
61004h–61007h	HBLANK_B	Pipe B Horizontal Blank	R/W
61008h–6100Bh	HSYNC_B	Pipe B Horizontal Sync	R/W
6100Ch–6100Fh	VTOTAL_B	Pipe B Vertical Total	R/W

Table 6-1. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
61010h–61013h	VBLANK_B	Pipe B Vertical Blank	R/W
61014h–61017h	VSYNC_B	Pipe B Vertical Sync	R/W
61018h–6101Bh	—	Reserved	—
6101Ch–6101Fh	PIPEBSRC	Pipe B Source Image Size	R/W
61020h–61023h	--	Reserved	--
61024h–61027h	—	Reserved	—
61028h–6102Bh	VSYNCSHIFT_B	Vertical Sync Shift Register B	—
6102Ch–6104Fh	—	Reserved	—
61050h–61053h	CRCCTRLREDB	Pipe B CRC Red Control	R/W
61054h–61057h	CRCCTRLGREENB	Pipe B CRC Green Control	R/W
61058h–6105Bh	CRCCTRLBLUEB	Pipe B CRC Blue Control	R/W
6105Ch–6105Fh	CRCCTRLRESB	Pipe B CRC Residual Control Register	R/W
61060h–61063h	CRCRESREDB	Pipe B CRC Red Result	RO
61064h–61067h	CRCRESGREENB	Pipe B CRC Green Result	RO
61068h–6106Bh	CRCRESBLUEB	Pipe B CRC Blue Result	RO
6106Ch–6106Fh	CRCRESRESB	Pipe B CRC Residual Result	RO
61070h–610FFh	—	Reserved	—
61100h–61103h	ADPA	Analog Display Port A Control	R/W
61104h–6110Fh	—	Reserved	—
61110h–61113h	PORT_HOTPLU_EN	Port HotPlug Enable	R/W
61114h–61117h	PORT_HOTPLU_STAT	Port HotPlug Status	R/W
61118h–61127h	—	Reserved	—
61128h–6112Bh	--	Reserved	--
6112Ch–6112Fh	—	Reserved	—
61130h–61133h	--	Reserved	--
61134h–61137h	--	Reserved	--
61138h–6113Bh	--	Reserved	--
6113Ch–6113Fh	—	Reserved	—
61140h-61143h	--	Reserved	--
61144h–61147h	--	Reserved	--
61148h–6114Bh	--	Reserved	--
6114Ch–6114Fh	--	Reserved	--
61150h-61153h	--	Reserved	--

Table 6-1. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
61154h-6115Fh	—	Reserved	—
61154h–61157h	--	Reserved	--
61158h–6115Bh	--	Reserved	--
6115Ch–6115Fh	--	Reserved	--
61160h-61163h	--	Reserved	--
61164h–61167h	--	Reserved	--
61168h–6116Bh	--	Reserved	--
6116Ch–6116Fh	—	Reserved	—
61170h–61173h	--	Reserved	--
61174h–61177h	—	Reserved	—
61178h–6117Bh	--	Reserved	--
6117Ch–61177h	—	Reserved	—
Reserved Registers (61400h–61FFFh)			
Reserved Registers (62000h–62FFFh)			
Reserved Registers (68000h–6FFFFh)			
Display and Cursor Control Registers (70000h–77FFFh)			
Display Pipeline A Control			
70000h–70003h	PIPEA_DSL	Pipe A Display Scan Line Count	RO
70004h–70007h	PIPEA_SLC	Pipe A Display Scan Line Count Range Compare	RO
70008h–7000Bh	PIPEACONF	Pipe A Configuration	R/W
7000Ch–7000Fh	—	Reserved	—
70010h–70013h	PIPEAGCMAXRED	Pipe A Gamma Correction Max Red	R/W
70014h–70017h	PIPEAGCMAXGRN	Pipe A Gamma Correction Max Green	R/W
70018h–7001Bh	PIPEAGCMAXBLU	Pipe A Gamma Correction Max Blue	R/W
7001Ch–70023h	—	Reserved	—
70024h–70027h	PIPEASTAT	Pipe A Display Status Select	R/W
70028h–7002Fh	—	Reserved	—
70030h–70033h	DSPARB	Display Arbitration Control	R/W
70034h–70037h	FW1	Display FIFO Watermark Control 1	R/W
70038h–7003Bh	FW2	Display FIFO Watermark Control 2	
7003Ch–7003Fh	FW3	Display FIFO Watermark Control 3	R/W
70040h-70043h	PIPEAFRAMEH	Pipe A Frame Count High	RO

Table 6-1. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
70044h-70047h	PIPEAFRAMEPIX	Pipe A Frame Count Low and Pixel Count	RO
70048h-7007Fh	—	Reserved	—
Cursor A & B Registers			
70080h–70083h	CURACNTR	Cursor A Control	R/W
70084h–70087h	CURABASE	Cursor A Base Address	R/W
70088h–7008Bh	CURAPOS	Cursor A Position	R/W
7008Ch–7008Fh	—	Reserved	—
70090h–7009Fh	CURAPALET[0:3]	Cursor A Palette 0:3	R/W
700A0h–700BFh	—	Reserved	—
700C0h–700C3h	CURBCNTR	Cursor B Control	R/W
700C4h–700C7h	CURBBASE	Cursor B Base Address	R/W
700C8h–700CBh	CURBPOS	Cursor B Position	R/W
700CCh–700CFh	—	Reserved	—
700D0h–700DFh	CURBPALET[0:3]	Cursor B Palette 0:3	R/W
700E0h–7017Fh	—	Reserved	—
Display A Control			
70180h–70183h	DSPACNTR	Display A Plane Control	R/W
70184h–70187h	DSPALINOFF	Display A Linear Offset Register	R/W
70188h–7018Bh	DSPASTRIDE	Display A Stride	R/W
7018Ch-7018Fh	—	Reserved	—
70190h-70193h	DSPARESV (RSVD)	Display A Reserved	R/W
70194h–70197h	DSPAKEYVAL	Sprite Color Key Value	R/W
70198h–7019Bh	DSPAKEYMSK	Sprite Color Key Mask Value	R/W
7019Ch–7019Fh	DSPASURF	Display A Surface Base Address Register	R/W
701A0h-701A3h	—	Reserved	—
701A4h–701A7h	DSPATILEOFF	Display A Tiled Offset Register	R/W
701A8h-701FFh	—	Reserved	—
70200h-70203h	DSPAFLPQSTAT	Flip Queue Status Register	R/W
70204h–703FFh	—	Reserved	—
VBIOS Software Flags 0-6			
70400h-70403h	--	Reserved	--
70404h–7040Fh	—	Reserved	—
70410h–7044Fh	SWF[00:0F]	Software Flag 00:0F	R/W

Table 6-1. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
70450h–70FFFh	—	Reserved	—
Display Pipeline B Control			
71000h–71003h	PIPEB_DSL	Pipe B Display Scan Line Count	RO
71004h–71007h	PIPEB_SLC	Pipe B Display Scan Line Range Compare	RO
71008h–7100Bh	PIPEBCONF	Pipe B Configuration	R/W
7100Ch–7100Fh	—	Reserved	—
71010h–71013h	PIPEBGCMAXRED	Pipe B Gamma Correction Max Red	R/W
71014h–71017h	PIPEBGCMAXGRN	Pipe B Gamma Correction Max Green	R/W
71018h–7101Bh	PIPEBGCMAXBLU	Pipe B Gamma Correction Max Blue	R/W
71024h–71027h	PIPEBSTAT	Pipe B Status	R/W
71028h–7103Fh	—	Reserved	—
71040h-71043h	PIPEBFRAMEH	Pipe B Frame Count High	RO
71044h-71047h	PIPEBFRAMEPIX	Pipe B Frame Count Low and Pixel Count	RO
71048h-7117Fh	—	Reserved	—
Display B / Sprite Control			
71180h–71183h	DSPBCNTR	Display B / Sprite Control	R/W
71184h–71187h	DSPBLINOFFSET	Display B / Sprite Linear Offset	R/W
71188h–7118Bh	DSPBSTRIDE	Display B / Sprite Stride	R/W
7118Ch–71193h	—	Reserved	—
71194h–71197h	DSPBKEYVAL	Display B / Sprite Color Key Value	R/W
71198h–7119Bh	DSPBKEYMSK	Display B / Sprite Color Key Mask	R/W
7119Ch–7119Fh	DSPBSURF	Display B Surface Base Address Register	R/W
711A0h-711A3h	—	Reserved	—
711A4h–711A7h	DSPBTILEOFF	Display B Tiled Offset Register	R/W
711A8h-711FFh	—	Reserved	—
71200h-71203h	DSPBFLPQSTAT	Flip Queue Status Register	R/W
71204h–713FFh	—	Reserved	—
Video BIOS Registers			
71400h–71403h	VGACNTRL	VGA Display Plane Control	R/W
71404h–7140Fh	—	Reserved	—
VBIOS Software Flags 10-1F			
71410h–7144Fh	SWF[10-1F]	Software Flag 10 – 1F	R/W

Table 6-1. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
71450h–71FFFh	—	Reserved	—
VBIOS Software Flags 30-32			
72400h–72413h	—	Reserved	—
72414h–72417h	SWF[30]	Software Flag 30	R/W
72418h–7241Bh	SWF[31]	Software Flag 31	R/W
7241Ch–7241Fh	SWF[32]	Software Flag 32	R/W
72420h–72FFFh	—	Reserved	—
Performance Counters (73000h-73FFFh)			
73000h–73003h	PCSRC	Performance Counter Source Register	R/W
73004h–73007h	PCSTAT	Performance Counter Status Register	RO
73008h–7317Fh	—	Reserved	—
Reserved (74000h-7FFFFh)			
74000h–7FFFFh	—	Reserved	—

6.2 VGA and Extended VGA Register Map

For I/O locations, the value in the address column represents the register I/O address. For memory mapped locations, this address is an offset from the base address programmed in the MMADR register.

6.2.1 VGA and Extended VGA I/O and Memory Register Map

Table 6-2. I/O and Memory Register Map

Address	Register Name (Read)	Register Name (Write)
2D Registers		
3B0h–3B3h	Reserved	Reserved
3B4h	VGA CRTC Index (CRX) (monochrome)	VGA CRTC Index (CRX) (monochrome)
3B5h	VGA CRTC Data (monochrome)	VGA CRTC Data (monochrome)
3B6h–3B9h	Reserved	Reserved
3Bah	VGA Status Register (ST01)	VGA Feature Control Register (FCR)
3BBh–3BFh	Reserved	Reserved
3C0h	VGA Attribute Controller Index (ARX)	VGA Attribute Controller Index (ARX)/ VGA Attribute Controller Data (alternating writes select ARX or write ARxx Data)

Address	Register Name (Read)	Register Name (Write)
3C1h	VGA Attribute Controller Data (read ARxx data)	Reserved
3C2h	VGA Feature Read Register (ST00)	VGA Miscellaneous Output Register (MSR)
3C3h	Reserved	Reserved
3C4h	VGA Sequencer Index (SRX)	VGA Sequencer Index (SRX)
3C5h	VGA Sequencer Data (SRxx)	VGA Sequencer Data (SRxx)
3C6h	VGA Color Palette Mask (DACMASK)	VGA Color Palette Mask (DACMASK)
3C7h	VGA Color Palette State (DACSTATE)	VGA Color Palette Read Mode Index (DACRX)
3C8h	VGA Color Palette Write Mode Index (DACWX)	VGA Color Palette Write Mode Index (DACWX)
3C9h	VGA Color Palette Data (DACDATA)	VGA Color Palette Data (DACDATA)
3CAh	VGA Feature Control Register (FCR)	Reserved
3CBh	Reserved	Reserved
3CCh	VGA Miscellaneous Output Register (MSR)	Reserved
3CDh	Reserved	Reserved
3CEh	VGA Graphics Controller Index (GRX)	VGA Graphics Controller Index (GRX)
3CFh	VGA Graphics Controller Data (GRxx)	VGA Graphics Controller Data (GRxx)
3D0h–3D1h	Reserved	Reserved
2D Registers		
3D4h	VGA CRTC Index (CRX)	VGA CRTC Index (CRX)
3D5h	VGA CRTC Data (CRxx)	VGA CRTC Data (CRxx)
System Configuration Registers		
3D6h	GFX/2D Configurations Extensions Index (XRX)	GFX/2D Configurations Extensions Index (XRX)
3D7h	GFX/2D Configurations Extensions Data (XRxx)	GFX/2D Configurations Extensions Data (XRxx)
2D Registers		
3D8h–3D9h	Reserved	Reserved
3DAh	VGA Status Register (ST01)	VGA Feature Control Register (FCR)
3DBh–3DFh	Reserved	Reserved

6.3 Indirect VGA and Extended VGA Register Indices

The registers listed in this section are indirectly accessed by programming an index value into the appropriate SRX, GRX, ARX, or CRX register. The index and data register address locations are listed in the previous section.

Table 6-3. 2D Sequence Registers (3C4h / 3C5h)

Index	Sym	Description
00h	SR00	Sequencer Reset
01h	SR01	Clocking Mode
02h	SR02	Plane / Map Mask
03h	SR03	Character Font
04h	SR04	Memory Mode
07h	SR07	Horizontal Character Counter Reset

Table 6-4. 2D Graphics Controller Registers (3CEh / 3CFh)

Index	Sym	Register Name
00h	GR00	Set / Reset
01h	GR01	Enable Set / Reset
02h	GR02	Color Compare
03h	GR03	Data Rotate
04h	GR04	Read Plane Select
05h	GR05	Graphics Mode
06h	GR06	Miscellaneous
07h	GR07	Color Don't Care
08h	GR08	Bit Mask
10h	GR10	Address Mapping
11h	GR11	Page Selector
18h	GR18	Software Flags

Table 6-5. 2D Attribute Controller Registers (3C0h / 3C1h)

Index	Sym	Register Name
00h	AR00	Palette Register 0
01h	AR01	Palette Register 1
02h	AR02	Palette Register 2
03h	AR03	Palette Register 3
04h	AR04	Palette Register 4
05h	AR05	Palette Register 5

Index	Sym	Register Name
06h	AR06	Palette Register 6
07h	AR07	Palette Register 7
08h	AR08	Palette Register 8
09h	AR09	Palette Register 9
0Ah	AR0A	Palette Register A
0Bh	AR0B	Palette Register B
0Ch	AR0C	Palette Register C
0Dh	AR0D	Palette Register D
0Eh	AR0E	Palette Register E
0Fh	AR0F	Palette Register F
10h	AR10	Mode Control
11h	AR11	Reserved
12h	AR12	Memory Plane Enable
13h	AR13	Horizontal Pixel Panning
14h	AR14	Color Select

Table 6-6. 2D CRT Controller Registers (3B4h / 3D4h / 3B5h / 3D5h)

Index	Sym	Register Name
00h	CR00	Horizontal Total
01h	CR01	Horizontal Display Enable End
02h	CR02	Horizontal Blanking Start
03h	CR03	Horizontal Blanking End
04h	CR04	Horizontal Sync Start
05h	CR05	Horizontal Sync End
06h	CR06	Vertical Total
07h	CR07	Overflow
08h	CR08	Preset Row Scan
09h	CR09	Maximum Scan Line
0Ah	CR0A	Text Cursor Start
0Bh	CR0B	Text Cursor End
0Ch	CR0C	Start Address High
0Dh	CR0D	Start Address Low
0Eh	CR0E	Text Cursor Location High
0Fh	CR0F	Text Cursor Location Low
10h	CR10	Vertical Sync Start
11h	CR11	Vertical Sync End

Index	Sym	Register Name
12h	CR12	Vertical Display Enable End
13h	CR13	Offset
14h	CR14	Underline Location
15h	CR15	Vertical Blanking Start
16h	CR16	Vertical Blanking End
17h	CR17	CRT Mode
18h	CR18	Line Compare
22h	CR22	Memory Read Latch Data
24h	CR24	Reserved

7. Memory Data Formats

This chapter describes the attributes associated with the memory-resident data objects operated on by the graphics pipeline. This includes object types, pixel formats, memory layouts, and rules/restrictions placed on the dimensions, physical memory location, pitch, alignment, etc. with respect to the specific operations performed on the objects.

7.1 Memory Object Overview

Any memory data accessed by the device is considered part of a *memory object* of some memory object type.

7.1.1 Memory Object Types

The following table lists the various memory objects types and an indication of their role in the system.

Memory Object Type	Role
Graphics Translation Table (GTT)	Contains PTEs used to translate "graphics addresses" into physical memory addresses.
Hardware Status Page	Cached page of sysmem used to provide fast driver synchronization.
Logical Context Buffer	Memory areas used to store (save/restore) images of hardware rendering contexts. Logical contexts are referenced via a pointer to the corresponding Logical Context Buffer.
Ring Buffers	Buffers used to transfer (DMA) instruction data to the device. Primary means of controlling rendering operations.
Batch Buffers	Buffers of instructions invoked indirectly from Ring Buffers.
State Descriptors	Contains state information in a prescribed layout format to be read by hardware. Many different state descriptor formats are supported.
Vertex Buffers	Buffers of 3D vertex data indirectly referenced through "indexed" 3D primitive instructions.
VGA Buffer (Must be mapped UC on PCI)	Graphics memory buffer used to drive the display output while in legacy VGA mode.
Display Surface	Memory buffer used to display images on display devices.
Overlay Surface	Memory buffer used to display overlaid images on display devices.
Overlay Register, Filter Coefficients Buffer	Memory area used to provide double-buffer for Overlay register and filter coefficient loading.
Cursor Surface	Hardware cursor pattern in memory.

Memory Object Type	Role
2D Render Source	Surface used as primary input to 2D rendering operations.
2D Render R-M-W Destination	2D rendering output surface that is read in order to be combined in the rendering function. Destination surfaces that accessed via this Read-Modify-Write mode have somewhat different restrictions than Write-Only Destination surfaces.
2D Render Write-Only Destination	2D rendering output surface that is written but not read by the 2D rendering function. Destination surfaces that accessed via a Write-Only mode have somewhat different restrictions than Read-Modify-Write Destination surfaces.
2D Monochrome Source	1 bpp surfaces used as inputs to 2D rendering after being converted to foreground/background colors.
2D Color Pattern	8x8 pixel array used to supply the "pattern" input to 2D rendering functions.
DIB	"Device Independent Bitmap" surface containing "logical" pixel values that are converted (via LUTs) to physical colors.
3D Color Buffer	Surface receiving color output of 3D rendering operations. May also be accessed via R-M-W (aka blending). Also referred to as a Render Target.
3D Depth Buffer	Surface used to hold per-pixel depth and stencil values used in 3D rendering operations. Accessed via RMW.
3D Texture Map	Color surface (or collection of surfaces) which provide texture data in 3D rendering operations.
"Non-3D" Texture	Surface read by Texture Samplers, though not in normal 3D rendering operations (e.g., in video color conversion functions).
Motion Comp Surfaces	These are the Motion Comp reference pictures.
Motion Comp Correction Data Buffer	This is Motion Comp intra-coded or inter-coded correction data.

7.2 Channel Formats

7.2.1 Unsigned Normalized (UNORM)

An unsigned normalized value with *n* bits is interpreted as a value between 0.0 and 1.0. The minimum value (all 0's) is interpreted as 0.0, the maximum value (all 1's) is interpreted as 1.0. Values inbetween are equally spaced. For example, a 2-bit UNORM value would have the four values 0, 1/3, 2/3, and 1.

If the incoming value is interpreted as an n-bit integer, the interpreted value can be calculated by dividing the integer by 2^n-1.

7.2.2 Gamma Conversion (SRGB)

Gamma conversion is only supported on UNORM formats. If this flag is included in the surface format name, it indicates that a reverse gamma conversion is to be done after the source surface is read, and a forward gamma conversion is to be done before the destination surface is written.

7.2.3 Signed Normalized (SNORM)

A signed normalized value with *n* bits is interpreted as a value between -1.0 and +1.0. If the incoming value is interpreted as a 2's-complement n-bit signed integer, the interpreted value can be calculated by dividing the integer by 2^{n-1}-1. Note that the most negative value of -2^{n-1} will result in a value slightly smaller than -1.0. This value is clamped to -1.0, thus there are two representations of -1.0 in SNORM format.

7.2.4 Unsigned Integer (UINT/USCALED)

The UINT and USCALED formats interpret the source as an unsigned integer value with *n* bits with a range
of 0 to 2^n-1.

The UINT formats copy the source value to the destination (zero-extending if required), keeping the value as an integer.

The USCALED formats convert the integer into the corresponding floating point value (e.g., 0x03 --> 3.0f). For 32-bit sources, the value is rounded to nearest even.

7.2.5 Signed Integer (SINT/SSCALED)

A signed integer value with *n* bits is interpreted as a 2's complement integer with a range of -2^{n-1} to $+2^{n-1}$-1.

The SINT formats copy the source value to the destination (sign-extending if required), keeping the value as an integer.

The SSCALED formats convert the integer into the corresponding floating point value (e.g., 0xFFFD --> -3.0f). For 32-bit sources, the value is rounded to nearest even.

7.2.6 Floating Point (FLOAT)

Refer to IEEE Standard 754 for Binary Floating-Point Arithmetic. The IA-32 Intel (R) Architecture Software Developer's Manual also describes floating point data types (though GEN deviates slightly from those behaviors).

7.2.6.1 32-bit Floating Point

Bit	Description
31	**Sign (s)**
30:23	**Exponent (e)** Biased Exponent
22:0	**Fraction (f)** Does not include "hidden one"

The value of this data type is derived as:

- if e == 255 and f != 0, then v is NaN regardless of s

- if e == 255 and f == 0, then v = $(-1)^s$*infinity (signed infinity)
- if 0 < e < 255, then v = $(-1)^s*2^{(e-127)}*(1.f)$
- if e == 0 and f != 0, then v = $(-1)^s*2^{(e-126)}*(0.f)$ (denormalized numbers)
- if e == 0 and f == 0, then v = $(-1)^s*0$ (signed zero)

7.2.6.2 64-bit Floating Point

Bit	Description
63	**Sign (s)**
62:52	**Exponent (e)** Biased Exponent
51:0	**Fraction (f)** Does not include “hidden one”

The value of this data type is derived as:

- if e == b’11..11’ and f != 0, then v is NaN regardless of s
- if e == b’11..11’ and f == 0, then v = $(-1)^s$*infinity (signed infinity)
- if 0 < e < b’11..11’, then v = $(-1)^s*2^{(e-1023)}*(1.f)$
- if e == 0 and f != 0, then v = $(-1)^s*2^{(e-1022)}*(0.f)$ (denormalized numbers)
- if e == 0 and f == 0, then v = $(-1)^s*0$ (signed zero)

7.2.6.3 16-bit Floating Point

Bit	Description
15	**Sign (s)**
14:10	**Exponent (e)** Biased Exponent
9:0	**Fraction (f)** Does not include “hidden one”

The value of this data type is derived as:

- if e == 31 and f != 0, then v is NaN regardless of s
- if e == 31 and f == 0, then v = $(-1)^s$*infinity (signed infinity)
- if 0 < e < 31, then v = $(-1)^s*2^{(e-15)}*(1.f)$
- if e == 0 and f != 0, then v = $(-1)^s*2^{(e-14)}*(0.f)$ (denormalized numbers)
- if e == 0 and f == 0, then v = $(-1)^s*0$ (signed zero)

The following table represents relationship between 32 bit and 16 bit floating point ranges:

flt32 exponent	Unbiased exponent		flt16 exponent	flt16 fraction
255				
254	127			
...				
127+16	16	Infinity	31	1.1111111111
127+15	15	Max exponent	30	1.xxxxxxxxxx
127	0		15	1.xxxxxxxxxx
113	-14	Min exponent	1	1.xxxxxxxxxx
112		Denormalized	0	0.1xxxxxxxxx
111		Denormalized	0	0.01xxxxxxxx
110		Denormalized	0	0.001xxxxxxx
109		Denormalized	0	0.0001xxxxxx
108		Denormalized	0	0.00001xxxxx
107		Denormalized	0	0.000001xxxx
106		Denormalized	0	0.0000001xxx
115		Denormalized	0	0.00000001xx
114		Denormalized	0	0.000000001x
113		Denormalized	0	0.0000000001
112		Denormalized	0	0.0
...				
0			0	0.0

Conversion from the 32-bit floating point format to the 16-bit format should be done with round to nearest even.

7.2.6.4 11-bit Floating Point

Bit	Description
10:6	**Exponent (e)** Biased Exponent
5:0	**Fraction (f)** Does not include “hidden one”

The value of this data type is derived as:

- if e == 31 and f != 0 then v = NaN
- if e == 31 and f == 0 then v = +infinity
- if 0 < e < 31, then v = $2^{(e-15)}$*(1.f)
- if e == 0 and f != 0, then v = $2^{(e-14)}$*(0.f) (denormalized numbers)
- if e == 0 and f == 0, then v = 0 (zero)

7.2.6.5 10-bit Floating Point

Bit	Description
9:5	**Exponent (e)** Biased Exponent
4:0	**Fraction (f)** Does not include “hidden one”

The value of this data type is derived as:

- if e == 31 and f != 0 then v = NaN
- if e == 31 and f == 0 then v = +infinity
- if 0 < e < 31, then v = $2^{(e-15)}$*(1.f)
- if e == 0 and f != 0, then v = $2^{(e-14)}$*(0.f) (denormalized numbers)
- if e == 0 and f == 0, then v = 0 (zero)

7.2.6.6 Shared Exponent

The R9G9B9E5_SHAREDEXP format contains three channels that share an exponent. The three fractions assume an impled “0” rather than an implied “1” as in the other floating point formats. This format does not support infinity and NaN values. There are no sign bits, only positive numbers and zero can be represented. The value of each channel is determined as follows, where “f” is the fraction of the corresponding channel, and “e” is the shared exponent.

$v = (0.f)*2^{(e-15)}$

Bit	Description
31:27	**Exponent (e)** Biased Exponent
26:18	**Blue Fraction**
17:9	**Green Fraction**
8:0	**Red Fraction**

7.3 Non-Video Surface Formats

This section describes the lowest-level organization of a surfaces containing discrete "pixel" oriented data (e.g., discrete pixel (RGB,YUV) colors, subsampled video data, 3D depth/stencil buffer pixel formats, bump map values etc. Many of these pixel formats are common to the various pixel-oriented memory object types.

7.3.1 Surface Format Naming

Unless indicated otherwise, all pixels are **stored** in "**little endian**" byte order. I.e., pixel bits 7:0 are stored in byte *n*, pixel bits 15:8 are stored in byte *n*+1, and so on. The format labels include color components in little endian order (e.g., R8G8B8A8 format is physically stored as R, G, B, A).

The name of most of the surface formats specifies its format. Channels are listed in little endian order (LSB channel on the left, MSB channel on the right), with the channel format specified following the channels with that format. For example, R5G5_SNORM_B6_UNORM contains, from LSB to MSB, 5 bits of red in SNORM format, 5 bits of green in SNORM format, and 6 bits of blue in UNORM format.

7.3.2 Intensity Formats

All surface formats containing "I" include an intensity value. When used as a source surface for the sampling engine, the intensity value is replicated to all four channels (R,G,B,A) before being filtered. Intensity surfaces are not supported as destinations.

7.3.3 Luminance Formats

All surface formats contaning "L" include a luminance value. When used as a source surface for the sampling engine, the luminance value is replicated to the three color channels (R,G,B) before being filtered. The alpha channel is provided either from another field or receives a default value. Luminance surfaces are not supported as destinations.

7.3.4 R1_UNORM (same as R1_UINT) and MONO8

When used as a texel format, the R1_UNORM format contains 8 1-bit Intensity (I) values that are replicated to all color channels. Note that T0 of byte 0 of a R1_UNORM-formatted texture corresponds to Texel[0,0]. This is different from the format used for monochrome sources in the Blt engine.

7	6	5	4	3	2	1	0
T7	T6	T5	T4	T3	T2	T1	T0

Bit	Description
T0	**Texel 0** On texture reads, this (unsigned) 1-bit value is replicated to all color channels. Format: U1
...	**...**
T7	**Texel 7** On texture reads, this (unsigned) 1-bit value is replicated to all color channels. Format: U1

MONO8 format is identical to R1_UNORM but has different semantics for filtering. MONO8 is the only supported format for the MAPFILTER_MONO filter. See the *Sampling Engine* chapter.

7.3.5 Palette Formats

7.3.5.1 P4A4_UNORM

This surface format contains a 4-bit Alpha value (in the high nibble) and a 4-bit Palette Index value (in the low nibble).

7 … 4	3 … 0
Alpha	Palette Index

Bit	Description
7:4	**Alpha** Alpha value which will be replicated to both the high and low nibble of an 8-bit value, and then divided by 255 to yield a [0.0,1.0] Alpha value. Format: U4
3:0	**Palette Index** A 4-bit index which is used to lookup a 24-bit (RGB) value in the texture palette (loaded via 3DSTATE_SAMPLER_PALETTE_LOADx) Format: U4

7.3.5.2 A4P4_UNORM

This surface format contains a 4-bit Alpha value (in the low nibble) and a 4-bit Color Index value (in the high nibble).

7 … 4	3 … 0
Palette Index	Alpha

Bit	Description
7:4	**Palette Index** A 4-bit color index which is used to lookup a 24-bit RGB value in the texture palette. Format: U4
3:0	**Alpha** Alpha value which will be replicated to both the high and low nibble of an 8-bit value, and then divided by 255 to yield a [0.0,1.0] alpha value. Format: U4

7.3.5.3 P8A8_UNORM

This surface format contains an 8-bit Alpha value (in the high byte) and an 8-bit Palette Index value (in the low byte).

15 … 8	7 … 0
Alpha	Palette Index

Bit	Description
7:4	**Alpha** Alpha value which will be divided by 255 to yield a [0.0,1.0] Alpha value. Format: U8
3:0	**Palette Index** An 8-bit index which is used to lookup a 24-bit (RGB) value in the texture palette (loaded via 3DSTATE_SAMPLER_PALETTE_LOADx) Format: U8

7.3.5.4 A8P8_UNORM

This surface format contains an 8-bit Alpha value (in the low byte) and an 8-bit Color Index value (in the high byte).

15 … 8	7 … 0
Palette Index	Alpha

Bit	Description
15:8	**Palette Index** An 8-bit color index which is used to lookup a 24-bit RGB value in the texture palette. Format: U8
7:0	**Alpha** Alpha value which will be divided by 255 to yield a [0.0,1.0] alpha value. Format: U8

7.3.5.5 P8_UNORM

This surface format contains only an 8-bit Color Index value.

Bit	Description
7:0	**Palette Index** An 8-bit color index which is used to lookup a 32-bit ARGB value in the texture palette. Format: U8

7.3.5.6 P2_UNORM

This surface format contains only a 2-bit Color Index value.

Bit	Description
1:0	**Palette Index** A 2-bit color index which is used to lookup a 32-bit ARGB value in the texture palette. Format: U2

7.4 Compressed Surface Formats

This section contains information on the internal organization of compressed surface formats.

7.4.1 FXT Texture Formats

There are four different FXT1 compressed texture formats. Each of the formats compress two 4x4 texel blocks into 128 bits. In each compression format, the 32 texels in the two 4x4 blocks are arranged according to the following diagram:

Figure 7-1. FXT1 Encoded Blocks

t0	t1	t2	t3
t4	t5	t6	t7
t8	t9	t10	t11
t12	t13	t14	t15

t16	t17	t18	t19
t20	t21	t22	t23
t24	t25	t26	t27
t28	t29	t30	t31

B6682-01

7.4.1.1 Overview of FXT1 Formats

During the compression phase, the encoder selects one of the four formats for each block based on which encoding scheme results in best overall visual quality. The following table lists the four different modes and their encodings:

Table 7-1. FXT1 Format Summary

Bit 127	Bit 126	Bit 125	Block Compression Mode	Summary Description
0	0	X	**CC_HI**	2 R5G5B5 colors supplied. Single LUT with 7 interpolated color values and transparent black
0	1	0	**CC_CHROMA**	4 R5G5B5 colors used directly as 4-entry LUT.
0	1	1	**CC_ALPHA**	3 A5R5G5B5 colors supplied. LERP bit selects between 1 LUT with 3 discrete colors + transparent black and 2 LUTs using interpolated values of Color 0,1 (t0-15) and Color 1,2 (t16-31).
1	x	x	**CC_MIXED**	4 R5G5B5 colors supplied, where Color0,1 LUT is used for t0-t15, and Color2,3 LUT used for t16-31. Alpha bit selects between LUTs with 4 interpolated colors or 3 interpolated colors + transparent black.

7.4.1.2 FXT1 CC_HI Format

In the CC_HI encoding format, two base 15-bit R5G5B5 colors (Color 0, Color 1) are included in the encoded block. These base colors are then expanded (using high-order bit replication) to 24-bit RGB colors, and used to define an 8-entry lookup table of interpolated color values (the 8th entry is transparent black). The encoded block contains a 3-bit index value per texel that is used to lookup a color from the table.

7.4.1.2.1 CC_HI Block Encoding

The following table describes the encoding of the 128-bit (DQWord) CC_HI block format:

Table 7-2. FXT CC_HI Block Encoding

Bit	Description
127:126	Mode = ‘00’b (CC_HI)
125:121	Color 1 Red
120:116	Color 1 Green
115:111	Color 1 Blue
110:106	Color 0 Red
105:101	Color 0 Green
100:96	Color 0 Blue
95:93	Texel 31 Select

Bit	Description
...	...
50:48	Texel 16 Select
47:45	Texel 15 Select
...	...
2:0	Texel 0 Select

7.4.1.2.2 CC_HI Block Decoding

The two base colors, Color 0 and Color 1 are converted from R5G5B5 to R8G8B8 by replicating the 3 MSBs into the 3 LSBs, as shown in the following table:

Table 7-3. FXT CC_HI Decoded Colors

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 1 [23:19]	Color 1 Red [7:3]	[125:121]
Color 1 [18:16]	Color 1 Red [2:0]	[125:123]
Color 1 [15:11]	Color 1 Green [7:3]	[120:116]
Color 1 [10:08]	Color 1 Green [2:0]	[120:118]
Color 1 [07:03]	Color 1 Blue [7:3]	[115:111]
Color 1 [02:00]	Color 1 Blue [2:0]	[115:113]
Color 0 [23:19]	Color 0 Red [7:3]	[110:106]
Color 0 [18:16]	Color 0 Red [2:0]	[110:108]
Color 0 [15:11]	Color 0 Green [7:3]	[105:101]
Color 0 [10:08]	Color 0 Green [2:0]	[105:103]
Color 0 [07:03]	Color 0 Blue [7:3]	[100:96]
Color 0 [02:00]	Color 0 Blue [2:0]	[100:98]

These two 24-bit colors (Color 0, Color 1) are then used to create a table of seven interpolated colors (with Alpha = 0FFh), along with an eight entry equal to RGBA = 0,0,0,0, as shown in the following table:

Table 7-4. FXT CC_HI Interpolated Color Table

Interpolated Color	Color RGB	Alpha
0	Color0.RGB	0FFh
1	(5 * Color0.RGB + 1 * Color1.RGB + 3) / 6	0FFh
2	(4 * Color0.RGB + 2 * Color1.RGB + 3) / 6	0FFh
3	(3 * Color0.RGB + 3 * Color1.RGB + 3) / 6	0FFh
4	(2 * Color0.RGB + 4 * Color1.RGB + 3) / 6	0FFh
5	(1 * Color0.RGB + 5 * Color1.RGB + 3) / 6	0FFh

Interpolated Color	Color RGB	Alpha
6	Color1.RGB	0FFh
7	RGB = 0,0,0	0

This table is then used as an 8-entry Lookup Table, where each 3-bit Texel n Select field of the encoded CC_HI block is used to index into a 32-bit A8R8G8B8 color from the table completing the decode of the CC_HI block.

7.4.1.3 FXT1 CC_CHROMA Format

In the CC_CHROMA encoding format, four 15-bit R5B5G5 colors are included in the encoded block. These colors are then expanded (using high-order bit replication) to form a 4-entry table of 24-bit RGB colors. The encoded block contains a 2-bit index value per texel that is used to lookup a 24-bit RGB color from the table. The Alpha component defaults to fully opaque (0FFh).

7.4.1.3.1 CC_CHROMA Block Encoding

The following table describes the encoding of the 128-bit (DQWord) CC_CHROMA block format:

Table 7-5. FXT CC_CHROMA Block Encoding

Bit	Description
127:125	Mode = ‘010’b (CC_CHROMA)
124	Unused
123:119	Color 3 Red
118:114	Color 3 Green
113:109	Color 3 Blue
108:104	Color 2 Red
103:99	Color 2 Green
98:94	Color 2 Blue
93:89	Color 1 Red
88:84	Color 1 Green
83:79	Color 1 Blue
78:74	Color 0 Red
73:69	Color 0 Green
68:64	Color 0 Blue
63:62	Texel 31 Select
...	
33:32	Texel 16 Select
31:30	Texel 15 Select

Bit	Description
...	
1:0	Texel 0 Select

7.4.1.3.2 CC_CHROMA Block Decoding

The four colors (Color 0-3) are converted from R5G5B5 to R8G8B8 by replicating the 3 MSBs into the 3 LSBs, as shown in the following tables:

Table 7-6. FXT CC_CHROMA Decoded Colors

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 3 [23:17]	Color 3 Red [7:3]	[123:119]
Color 3 [18:16]	Color 3 Red [2:0]	[123:121]
Color 3 [15:11]	Color 3 Green [7:3]	[118:114]
Color 3 [10:08]	Color 3 Green [2:0]	[118:116]
Color 3 [07:03]	Color 3 Blue [7:3]	[113:109]
Color 3 [02:00]	Color 3 Blue [2:0]	[113:111]
Color 2 [23:17]	Color 2 Red [7:3]	[108:104]
Color 2 [18:16]	Color 2 Red [2:0]	[108:106]
Color 2 [15:11]	Color 2 Green [7:3]	[103:99]
Color 2 [10:08]	Color 2 Green [2:0]	[103:101]
Color 2 [07:03]	Color 2 Blue [7:3]	[98:94]
Color 2 [02:00]	Color 2 Blue [2:0]	[98:96]
Color 1 [23:17]	Color 1 Red [7:3]	[93:89]
Color 1 [18:16]	Color 1 Red [2:0]	[93:91]
Color 1 [15:11]	Color 1 Green [7:3]	[88:84]
Color 1 [10:08]	Color 1 Green [2:0]	[88:86]
Color 1 [07:03]	Color 1 Blue [7:3]	[83:79]
Color 1 [02:00]	Color 1 Blue [2:0]	[83:81]
Color 0 [23:17]	Color 0 Red [7:3]	[78:74]
Color 0 [18:16]	Color 0 Red [2:0]	[78:76]
Color 0 [15:11]	Color 0 Green [7:3]	[73:69]
Color 0 [10:08]	Color 0 Green [2:0]	[73:71]
Color 0 [07:03]	Color 0 Blue [7:3]	[68:64]
Color 0 [02:00]	Color 0 Blue [2:0]	[68:66]

This table is then used as a 4-entry Lookup Table, where each 2-bit Texel n Select field of the encoded CC_CHROMA block is used to index into a 32-bit A8R8G8B8 color from the table (Alpha defaults to 0FFh) completing the decode of the CC_CHROMA block.

Table 7-7. FXT CC_CHROMA Interpolated Color Table

Texel Select	Color ARGB
0	Color0.ARGB
1	Color1.ARGB
2	Color2.ARGB
3	Color3.ARGB

7.4.1.4 FXT1 CC_MIXED Format

In the CC_MIXED encoding format, four 15-bit R5G5B5 colors are included in the encoded block: Color 0 and Color 1 are used for Texels 0-15, and Color 2 and Color 3 are used for Texels 16-31.

Each pair of colors are then expanded (using high-order bit replication) to form 4-entry tables of 24-bit RGB colors. The encoded block contains a 2-bit index value per texel that is used to lookup a 24-bit RGB color from the table. The Alpha component defaults to fully opaque (0FFh).

7.4.1.4.1 CC_MIXED Block Encoding

The following table describes the encoding of the 128-bit (DQWord) CC_MIXED block format:

Table 7-8. FXT CC_MIXED Block Encoding

Bit	Description
127	Mode = ‘1’b (CC_MIXED)
126	Color 3 Green [0]
125	Color 1 Green [0]
124	Alpha [0]
123:119	Color 3 Red
118:114	Color 3 Green
113:109	Color 3 Blue
108:104	Color 2 Red
103:99	Color 2 Green
98:94	Color 2 Blue
93:89	Color 1 Red
88:84	Color 1 Green
83:79	Color 1 Blue
78:74	Color 0 Red
73:69	Color 0 Green
68:64	Color 0 Blue

Bit	Description
63:62	Texel 31 Select
...	...
33:32	Texel 16 Select
31:30	Texel 15 Select
...	...
1:0	Texel 0 Select

7.4.1.4.2 CC_MIXED Block Decoding

The decode of the CC_MIXED block is modified by Bit 124 (Alpha [0]) of the encoded block.

Alpha[0] = 0 Decoding

When Alpha[0] = 0 the four colors are encoded as 16-bit R5G6B5 values, with the Green LSB defined as per the following table:

Table 7-9. FXT CC_MIXED (Alpha[0]=0) Decoded Colors

Encoded Color Bit	Definition
Color 3 Green [0]	Encoded Bit [126]
Color 2 Green [0]	Encoded Bit [33] XOR Encoded Bit [126]
Color 1 Green [0]	Encoded Bit [125]
Color 0 Green [0]	Encoded Bit [1] XOR Encoded Bit [125]

The four colors (Color 0-3) are then converted from R5G5B6 to R8G8B8 by replicating the 3 MSBs into the 3 LSBs, as shown in the following table:

Table 7-10. FXT CC_MIXED Decoded Colors (Alpha[0] = 0)

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 3 [23:17]	Color 3 Red [7:3]	[123:119]
Color 3 [18:16]	Color 3 Red [2:0]	[123:121]
Color 3 [15:11]	Color 3 Green [7:3]	[118:114]
Color 3 [10]	Color 3 Green [2]	[126]
Color 3 [09:08]	Color 3 Green [1:0]	[118:117]
Color 3 [07:03]	Color 3 Blue [7:3]	[113:109]
Color 3 [02:00]	Color 3 Blue [2:0]	[113:111]
Color 2 [23:17]	Color 2 Red [7:3]	[108:104]
Color 2 [18:16]	Color 2 Red [2:0]	[108:106]
Color 2 [15:11]	Color 2 Green [7:3]	[103:99]
Color 2 [10]	Color 2 Green [2]	[33] XOR [126]]

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 2 [09:08]	Color 2 Green [1:0]	[103:100]
Color 2 [07:03]	Color 2 Blue [7:3]	[98:94]
Color 2 [02:00]	Color 2 Blue [2:0]	[98:96]
Color 1 [23:17]	Color 1 Red [7:3]	[93:89]
Color 1 [18:16]	Color 1 Red [2:0]	[93:91]
Color 1 [15:11]	Color 1 Green [7:3]	[88:84]
Color 1 [10]	Color 1 Green [2]	[125]
Color 1 [09:08]	Color 1 Green [1:0]	[88:86]
Color 1 [07:03]	Color 1 Blue [7:3]	[83:79]
Color 1 [02:00]	Color 1 Blue [2:0]	[83:81]
Color 0 [23:17]	Color 0 Red [7:3]	[78:74]
Color 0 [18:16]	Color 0 Red [2:0]	[78:76]
Color 0 [15:11]	Color 0 Green [7:3]	[73:69]
Color 0 [10]	Color 0 Green [2]	[1] XOR [125]
Color 0 [09:08]	Color 0 Green [1:0]	[73:71]
Color 0 [07:03]	Color 0 Blue [7:3]	[68:64]
Color 0 [02:00]	Color 0 Blue [2:0]	[68:66]

The two sets of 24-bit colors (Color 0,1 and Color 2,3) are then used to create two tables of four interpolated colors (with Alpha = 0FFh). The Color0,1 table is used as a lookup table for texel 0-15 indices, and the Color2,3 table used for texels 16-31 indices, as shown in the following figures:

Table 7-11. FXT CC_MIXED Interpolated Color Table (Alpha[0]=0, Texels 0-15)

Texel 0-15 Select	Color RGB	Alpha
0	Color0.RGB	0FFh
1	(2*Color0.RGB + Color1.RGB + 1) /3	0FFh
2	(Color0.RGB + 2*Color1.RGB + 1) /3	0FFh
3	Color1.RGB	0FFh

Table 7-12. FXT CC_MIXED Interpolated Color Table (Alpha[0]=0, Texels 16-31)

Texel 16-31 Select	Color RGB	Alpha
0	Color2.RGB	0FFh
1	(2/3) * Color2.RGB + (1/3) * Color3.RGB	0FFh
2	(1/3) * Color2.RGB + (2/3) * Color3.RGB	0FFh
3	Color3.RGB	0FFh

Alpha[0] = 1 Decoding

When Alpha[0] = 1, Color0 and Color2 are encoded as 15-bit R5G5B5 values. Color1 and Color3 are encoded as RGB565 colors, with the Green LSB obtained as shown in the following table:

Table 7-13. FXT CC_MIXED (Alpha[0]=0) Decoded Colors

Encoded Color Bit	Definition
Color 3 Green [0]	Encoded Bit [126]
Color 1 Green [0]	Encoded Bit [125]

All four colors are then expanded to 24-bit R8G8B8 colors by bit replication, as show in the following diagram.

Table 7-14. FXT CC_MIXED Decoded Colors (Alpha[0] = 1)

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 3 [23:17]	Color 3 Red [7:3]	[123:119]
Color 3 [18:16]	Color 3 Red [2:0]	[123:121]
Color 3 [15:11]	Color 3 Green [7:3]	[118:114]
Color 3 [10]	Color 3 Green [2]	[126]
Color 3 [09:08]	Color 3 Green [1:0]	[118:117]
Color 3 [07:03]	Color 3 Blue [7:3]	[113:109]
Color 3 [02:00]	Color 3 Blue [2:0]	[113:111]
Color 2 [23:19]	Color 2 Red [7:3]	[108:104]
Color 2 [18:16]	Color 2 Red [2:0]	[108:106]
Color 2 [15:11]	Color 2 Green [7:3]	[103:99]
Color 2 [10:08]	Color 2 Green [2:0]	[103:101]
Color 2 [07:03]	Color 2 Blue [7:3]	[98:94]
Color 2 [02:00]	Color 2 Blue [2:0]	[98:96]
Color 1 [23:17]	Color 1 Red [7:3]	[93:89]
Color 1 [18:16]	Color 1 Red [2:0]	[93:91]
Color 1 [15:11]	Color 1 Green [7:3]	[88:84]

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 1 [10]	Color 1 Green [2]	[125]
Color 1 [09:08]	Color 1 Green [1:0]	[88:87]
Color 1 [07:03]	Color 1 Blue [7:3]	[83:79]
Color 1 [02:00]	Color 1 Blue [2:0]	[83:81]
Color 0 [23:19]	Color 0 Red [7:3]	[78:74]
Color 0 [18:16]	Color 0 Red [2:0]	[78:76]
Color 0 [15:11]	Color 0 Green [7:3]	[73:69]
Color 0 [10:08]	Color 0 Green [2:0]	[73:71]
Color 0 [07:03]	Color 0 Blue [7:3]	[68:64]
Color 0 [02:00]	Color 0 Blue [2:0]	[68:66]

The two sets of 24-bit colors (Color 0,1 and Color 2,3) are then used to create two tables of four colors. The Color0,1 table is used as a lookup table for texel 0-15 indices, and the Color2,3 table used for texels 16-31 indices. The color at index 1 is the linear interpolation of the base colors, while the color at index 3 is defined as Black (0,0,0) with Alpha = 0, as shown in the following figures:

Table 7-15. FXT CC_MIXED Interpolated Color Table (Alpha[0]=1, Texels 0-15)

Texel 0-15 Select	Color RGB	Alpha
0	Color0.RGB	0FFh
1	(Color0.RGB + Color1.RGB) /2	0FFh
2	Color1.RGB	0FFh
3	Black (0,0,0)	0

Table 7-16. FXT CC_MIXED Interpolated Color Table (Alpha[0]=1, Texels 16-31)

Texel 16-31 Select	Color RGB	Alpha
0	Color2.RGB	0FFh
1	(Color2.RGB + Color3.RGB) /2	0FFh
2	Color3.RGB	0FFh
3	Black (0,0,0)	0

These tables are then used as a 4-entry Lookup Table, where each 2-bit Texel n Select field of the encoded CC_MIXED block is used to index into the appropriate 32-bit A8R8G8B8 color from the table, completing the decode of the CC_CMIXED block.

7.4.1.5 FXT1 CC_ALPHA Format

In the CC_ALPHA encoding format, three A5R5G5B5 colors are provided in the encoded block. A control bit (LERP) is used to define the lookup table (or tables) used to dereference the 2-bit Texel Selects.

7.4.1.5.1 CC_ALPHA Block Encoding

The following table describes the encoding of the 128-bit (DQWord) CC_ALPHA block format:

Table 7-17. FXT CC_ALPHA Block Encoding

Bit	Description
127:125	Mode = ‘011’b (CC_ALPHA)
124	LERP
123:119	Color 2 Alpha
118:114	Color 1 Alpha
113:109	Color 0 Alpha
108:104	Color 2 Red
103:99	Color 2 Green
98:94	Color 2 Blue
93:89	Color 1 Red
88:84	Color 1 Green
83:79	Color 1 Blue
78:74	Color 0 Red
73:69	Color 0 Green
68:64	Color 0 Blue
63:62	Texel 31 Select
...	...
33:32	Texel 16 Select
31:30	Texel 15 Select
...	...
1:0	Texel 0 Select

7.4.1.5.2 CC_ALPHA Block Decoding

Each of the three colors (Color 0-2) are converted from A5R5G5B5 to A8R8G8B8 by replicating the 3 MSBs into the 3 LSBs, as shown in the following tables:

Table 7-18. FXT CC_ALPHA Decoded Colors

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 2 [31:27]	Color 2 Alpha [7:3]	[123:119]
Color 2 [26:24]	Color 2 Alpha [2:0]	[123:121]
Color 2 [23:17]	Color 2 Red [7:3]	[108:104]
Color 2 [18:16]	Color 2 Red [2:0]	[108:106]
Color 2 [15:11]	Color 2 Green [7:3]	[103:99]
Color 2 [10:08]	Color 2 Green [2:0]	[103:101]
Color 2 [07:03]	Color 2 Blue [7:3]	[98:94]
Color 2 [02:00]	Color 2 Blue [2:0]	[98:96]
Color 1 [31:27]	Color 1 Alpha [7:3]	[118:114]
Color 1 [26:24]	Color 1 Alpha [2:0]	[118:116]
Color 1 [23:17]	Color 1 Red [7:3]	[93:89]
Color 1 [18:16]	Color 1 Red [2:0]	[93:91]
Color 1 [15:11]	Color 1 Green [7:3]	[88:84]
Color 1 [10:08]	Color 1 Green [2:0]	[88:86]
Color 1 [07:03]	Color 1 Blue [7:3]	[83:79]
Color 1 [02:00]	Color 1 Blue [2:0]	[83:81]
Color 0 [31:27]	Color 0 Alpha [7:3]	[113:109]
Color 0 [26:24]	Color 0 Alpha [2:0]	[113:111]
Color 0 [23:17]	Color 0 Red [7:3]	[78:74]
Color 0 [18:16]	Color 0 Red [2:0]	[78:76]
Color 0 [15:11]	Color 0 Green [7:3]	[73:69]
Color 0 [10:08]	Color 0 Green [2:0]	[73:71]
Color 0 [07:03]	Color 0 Blue [7:3]	[68:64]
Color 0 [02:00]	Color 0 Blue [2:0]	[68:66]

LERP = 0 Decoding

When LERP = 0, a single 4-entry lookup table is formed using the three expanded colors, with the 4th entry defined as transparent black (ARGB=0,0,0,0). Each 2-bit Texel n Select field of the encoded CC_ALPHA block is used to index into a 32-bit A8R8G8B8 color from the table completing the decode of the CC_ALPHA block.

Table 7-19. FXT CC_ALPHA Interpolated Color Table (LERP=0)

Texel Select	Color	Alpha
0	Color0.RGB	Color0.Alpha
1	Color1.RGB	Color1.Alpha
2	Color2.RGB	Color2.Alpha
3	Black (RGB=0,0,0)	0

LERP = 1 Decoding

When LERP = 1, the three expanded colors are used to create two tables of four interpolated colors. The Color0,1 table is used as a lookup table for texel 0-15 indices, and the Color1,2 table used for texels 16-31 indices, as shown in the following figures:

Table 7-20. FXT CC_ALPHA Interpolated Color Table (LERP=1, Texels 0-15)

Texel 0-15 Select	Color ARGB
0	Color0.ARGB
1	(2*Color0.ARGB + Color1.ARGB + 1) /3
2	(Color0.ARGB + 2*Color1.ARGB + 1) /3
3	Color1.ARGB

Table 7-21. FXT CC_ALPHA Interpolated Color Table (LERP=1, Texels 16-31)

Texel 16-31 Select	Color ARGB
0	Color2.ARGB
1	(2*Color2.ARGB + Color1.ARGB + 1) /3
2	(Color2.ARGB + 2*Color1.ARGB + 1) /3
3	Color1.ARGB

7.4.2 BC4

These formats (BC4_UNORM and BC4_SNORM) compresses single-component UNORM or SNORM data. An 8-byte compression block represents a 4x4 block of texels. The texels are labeled as texel[row][column] where both row and column range from 0 to 3. Texel[0][0] is the upper left texel.

The 8-byte compression block is laid out as follows:

Bit	Description
7:0	red_0
15:8	red_1
18:16	texel[0][0] bit code
21:19	texel[0][1] bit code
24:22	texel[0][2] bit code
27:25	texel[0][3] bit code
30:28	texel[1][0] bit code
33:31	texel[1][1] bit code
36:34	texel[1][2] bit code
39:37	texel[1][3] bit code
42:40	texel[2][0] bit code
45:43	texel[2][1] bit code
48:46	texel[2][2] bit code
51:49	texel[2][3] bit code
54:52	texel[3][0] bit code
57:55	texel[3][1] bit code
60:58	texel[3][2] bit code
63:61	texel[3][3] bit code

There are two interpolation modes, chosen based on which reference color is larger. The first mode has the two reference colors plus six equal-spaced interpolated colors between the reference colors, chosen based on the three-bit code for that texel. The second mode has the two reference colors plus four interpolated colors, chosen by six of the three-bit codes. The remaining two codes select min and max values for the colors. The values of red_0 through red_7 are computed as follows:

```
red_0 = red_0;                              // bit code 000

red_1 = red_1;                              // bit code 001

if (red_0 > red_1)

{

        red_2 = (6 * red_0 + 1 * red_1) / 7;   // bit code 010

        red_3 = (5 * red_0 + 2 * red_1) / 7;   // bit code 011

        red_4 = (4 * red_0 + 3 * red_1) / 7;   // bit code 100

        red_5 = (3 * red_0 + 4 * red_1) / 7;   // bit code 101

        red_6 = (2 * red_0 + 5 * red_1) / 7;   // bit code 110

        red_7 = (1 * red_0 + 6 * red_1) / 7;   // bit code 111

}

else

{

        red_2 = (4 * red_0 + 1 * red_1) / 5;   // bit code 010

        red_3 = (3 * red_0 + 2 * red_1) / 5;   // bit code 011

        red_4 = (2 * red_0 + 3 * red_1) / 5;   // bit code 100

        red_5 = (1 * red_0 + 4 * red_1) / 5;   // bit code 101

        red_6 = UNORM ? 0.0 : -1.0;            // bit code 110 (0 for UNORM, -1 for SNORM)

        red_7 = 1.0;                            // bit code 111

}
```

7.4.3 BC5

These formats (BC5_UNORM and BC5_SNORM) compresses dual-component UNORM or SNORM data. A 16-byte compression block represents a 4x4 block of texels. The texels are labeled as texel[row][column] where both row and column range from 0 to 3. Texel[0][0] is the upper left texel.

The 16-byte compression block is laid out as follows:

Bit	Description
7:0	red_0
15:8	red_1
18:16	texel[0][0] red bit code
21:19	texel[0][1] red bit code
24:22	texel[0][2] red bit code
27:25	texel[0][3] red bit code
30:28	texel[1][0] red bit code
33:31	texel[1][1] red bit code
36:34	texel[1][2] red bit code
39:37	texel[1][3] red bit code
42:40	texel[2][0] red bit code
45:43	texel[2][1] red bit code
48:46	texel[2][2] red bit code
51:49	texel[2][3] red bit code
54:52	texel[3][0] red bit code
57:55	texel[3][1] red bit code
60:58	texel[3][2] red bit code
63:61	texel[3][3] red bit code
71:64	green_0
79:72	green_1
82:80	texel[0][0] green bit code
85:83	texel[0][1] green bit code
88:86	texel[0][2] green bit code
91:89	texel[0][3] green bit code
94:92	texel[1][0] green bit code
97:95	texel[1][1] green bit code
100:98	texel[1][2] green bit code
103:101	texel[1][3] green bit code
106:104	texel[2][0] green bit code
109:107	texel[2][1] green bit code
112:110	texel[2][2] green bit code
115:113	texel[2][3] green bit code
118:116	texel[3][0] green bit code
121:119	texel[3][1] green bit code
124:122	texel[3][2] green bit code
127:125	texel[3][3] green bit code

There are two interpolation modes, chosen based on which reference color is larger. The first mode has the two reference colors plus six equal-spaced interpolated colors between the reference colors, chosen based on the three-bit code for that texel. The second mode has the two reference colors plus four

interpolated colors, chosen by six of the three-bit codes. The remaining two codes select min and max values for the colors. The values of red_0 through red_7 are computed as follows:

```
red_0 = red_0;                              // bit code 000

red_1 = red_1;                              // bit code 001

if (red_0 > red_1)

{

        red_2 = (6 * red_0 + 1 * red_1) / 7;   // bit code 010

        red_3 = (5 * red_0 + 2 * red_1) / 7;   // bit code 011

        red_4 = (4 * red_0 + 3 * red_1) / 7;   // bit code 100

        red_5 = (3 * red_0 + 4 * red_1) / 7;   // bit code 101

        red_6 = (2 * red_0 + 5 * red_1) / 7;   // bit code 110

        red_7 = (1 * red_0 + 6 * red_1) / 7;   // bit code 111

}

else

{

        red_2 = (4 * red_0 + 1 * red_1) / 5;   // bit code 010

        red_3 = (3 * red_0 + 2 * red_1) / 5;   // bit code 011

        red_4 = (2 * red_0 + 3 * red_1) / 5;   // bit code 100

        red_5 = (1 * red_0 + 4 * red_1) / 5;   // bit code 101

        red_6 = UNORM ? 0.0 : -1.0;            // bit code 110 (0 for UNORM, -1 for SNORM)

        red_7 = 1.0;                            // bit code 111

}
```

The same calculations are done for green, using the corresponding reference colors and bit codes.

7.5 Video Pixel/Texel Formats

This section describes the “video” pixel/texel formats with respect to memory layout. See the Overlay chapter for a description of how the Y, U, V components are sampled.

7.5.1 Packed Memory Organization

Color components are all 8 bits in size for YUV formats. For YUV 4:2:2 formats each DWord will contain two pixels and only the byte order affects the memory organization.

The following four YUV 4:2:2 surface formats are supported, listed with alternate names:

- YCRCB_NORMAL (YUYV/YUY2)
- YCRCB_SWAPUVY (VYUY) (R8G8_B8G8_UNORM)
- YCRCB_SWAPUV (YVYU) (G8R8_G8B8_UNORM)
- YCRCB_SWAPY (UYVY)

The channels are mapped as follows:

Cr (V)	Red
Y	Green
Cb (U)	Blue

Figure 7-2. Memory layout of packed YUV 4:2:2 formats

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
V								Y								U								Y							
Pixel N								Pixel N+1								Pixel N															

YUV 4:2:2 (Normal)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U								Y								V								Y							
Pixel N								Pixel N+1								Pixel N															

YUV 4:2:2 (UV Swap)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Y								V								Y								U							
Pixel N+1								Pixel N								Pixel N															

YUV 4:2:2 (Y Swap)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Y								U								Y								V							
Pixel N+1								Pixel N								Pixel N															

YUV 4:2:2 (UV/Y Swap)

B6683-01

7.5.2 Planar Memory Organization

Planar formats use what could be thought of as separate buffers for the three color components. Because there is a separate stride for the Y and U/V data buffers, several memory footprints can be supported.

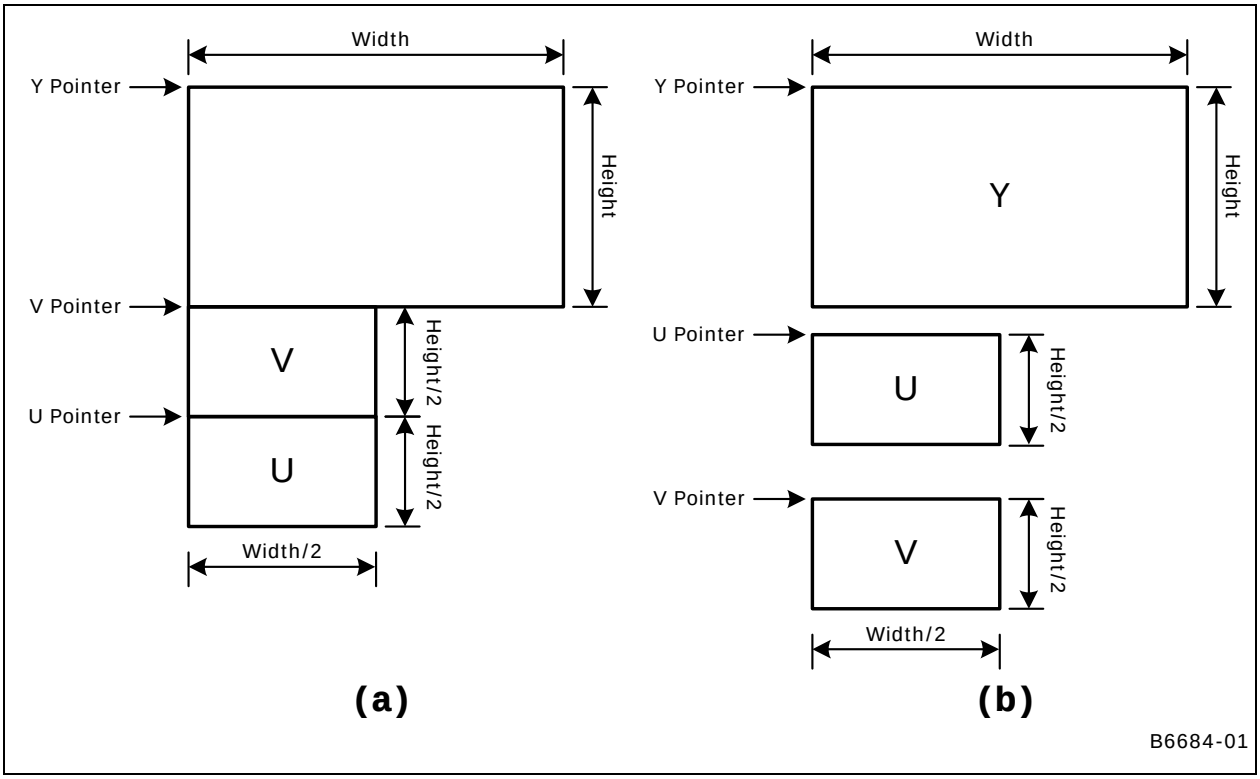
Note: There is no direct support for use of planar video surfaces as textures. The sampling engine can be used to operate on each of the 8bpp buffers separately (via a single-channel 8-bit format such as I8_UNORM). The U and V buffers can be written concurrently by using multiple render targets from the pixel shader. The Y buffer must be written in a separate pass due to its different size.

The following figure shows two types of memory organization for the YUV 4:2:0 planar video data:

1. The memory organization of the common YV12 data, where all three planes are contiguous and the strides of U and V components are half of that of the Y component.

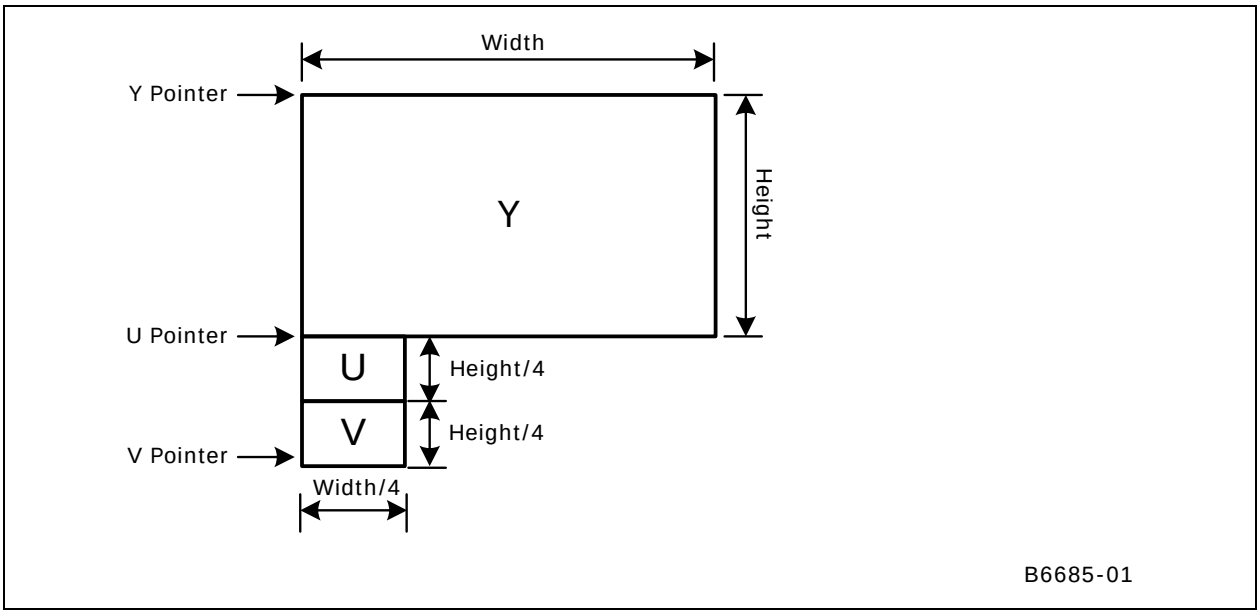
2. An alternative memory structure that the addresses of the three planes are independent but satisfy certain alignment restrictions.

Figure 7-3. YUV 4:2:0 Format Memory Organization



The following figure shows memory organization of the planar YUV 4:1:0 format where the planes are contiguous. The stride of the U and V planes is a quarter of that of the Y plane.

Figure 7-4. YUV 4:1:0 Format Memory Organization



7.6 Surface Memory Organizations

See *Memory Interface Functions* chapter for a discussion of tiled vs. linear surface formats.

7.7 Graphics Translation Tables

The Graphics Translation Tables GTT (Graphics Translation Table, sometimes known as the global GTT) and PPGTT (Per-Process Graphics Translation Table) are memory-resident page tables containing an array of DWord Page Translation Entries (PTEs) used in mapping logical Graphics Memory addresses to physical memory addresses, and sometimes snooped system memory "PCI" addresses.

The graphics translation tables must reside in (unsnooped) system memory.

The base address (MM offset) of the GTT and the PPGTT are programmed via the PGTBL_CTL and PGTBL_CTL2 MI registers, respectively. The translation table base addresses must be 4KB aligned. The GTT size can be either 128KB, 256KB or 512KB (mapping to 128MB, 256MB, and 512MB aperture sizes respectively) and is physically contiguous. The global GTT should only be programmed via the range defined by GTTADR. The PPGTT is programmed directly in memory. The per-process GTT (PPGTT) size is controlled by the PGTBL_CTL2 register. The PPGTT can, in addition to the above sizes, also be 64KB in size (corresponding to a 64MB aperture). Refer to the GTT Range chapter for a bit definition of the PTE entries.

7.8 Hardware Status Page

The hardware status page is a naturally-aligned 4KB page residing in snooped system memory. This page exists primarily to allow the device to report status via PCI master writes – thereby allowing the driver to read/poll WB memory instead of UC reads of device registers or UC memory.

The address of this page is programmed via the HWS_PGA MI register. The definition of that register (in *Memory Interface Registers*) includes a description of the layout of the Hardware Status Page.

7.9 Instruction Ring Buffers

Instruction ring buffers are the memory areas used to pass instructions to the device. Refer to the *Programming Interface* chapter for a description of how these buffers are used to transport instructions.

The RINGBUF register sets (defined in *Memory Interface Registers*) are used to specify the ring buffer memory areas. The ring buffer must start on a 4KB boundary and be allocated in linear memory. The length of any one ring buffer is limited to 2MB.

Note that "indirect" 3D primitive instructions (those that access vertex buffers) must reside in the same memory space as the vertex buffers.

7.10 Instruction Batch Buffers

Instruction batch buffers are contiguous streams of instructions referenced via an MI_BATCH_BUFFER_START and related instructions (see Memory Interface Instructions, Programming Interface). They are used to transport instructions external to ring buffers.

Note that batch buffers should not be mapped to snooped SM (PCI) addresses. The device will treat these as MainMemory (MM) address, and therefore not snoop the CPU cache.

The batch buffer must be QWord aligned and a multiple of QWords in length. The ending address is the address of the last valid QWord in the buffer. The length of any single batch buffer is "virtually unlimited" (i.e., could theoretically be 4GB in length).

7.11 Display, Overlay, Cursor Surfaces

These surfaces are memory image buffers (planes) used to refresh a display device in non-VGA mode. See the *Display* chapter for specifics on how these surfaces are defined/used.

7.12 2D Render Surfaces

These surfaces are used as general source and/or destination operands in 2D Blt operations.

Note that the device provides no coherency between 2D render surfaces and the texture cache – i.e., the texture cache must be explicitly invalidated prior to the use of a texture that has been modified via the Blt engine.

See the *2D Instruction* and *2D Rendering* chapters for specifics on how these surfaces are used, restrictions on their size, placement, etc.

7.13 2D Monochrome Source

These 1bpp surfaces are used as source operands to certain 2D Blt operations, where the Blt engine expands the 1bpp source into the required color depth.

The device uses the texture cache to store monochrome sources. There is no mechanism to maintain coherency between 2D render surfaces and (texture)-cached monochrome sources, software is required to explicitly invalidate the texture cache before using a memory-based monochrome source that has been modified via the Blt engine. (Here the assumption is that SW enforces memory-based monochrome source surfaces as read-only surfaces).

See the *2D Instruction* and *2D Rendering* chapters for specifics on how these surfaces are used, restrictions on their size, placement, coherency rules, etc.

7.14 2D Color Pattern

Color pattern surfaces are used as special pattern operands in 2D Blt operations.

The device uses the texture cache to store color patterns. There is no mechanism to maintain coherency between 2D render surfaces and (texture)-cached color patterns, software is required to explicitly invalidate the texture cache before using a memory-based color pattern that has been modified via the Blt engine. (Here the assumption is that SW enforces memory-based color pattern surfaces as read-only surfaces).

See the *2D Instruction* and *2D Rendering* chapters for specifics on how these surfaces are used, restrictions on their size, placement, etc.

7.15 3D Color Buffer (Destination) Surfaces

3D Color buffer surfaces are used to hold per-pixel color values for use in the 3D pipeline. Note that the 3D pipeline <u>always requires</u> a Color buffer to be defined.

Refer to Non-Video Pixel/Texel Formats section in this chapter for details on the Color buffer pixel formats. Refer to the *3D Instruction* and *3D Rendering chapters* for details on the usage of the Color Buffer.

The Color buffer is defined as the BUFFERID_COLOR_BACK memory buffer via the 3DSTATE_BUFFER_INFO instruction. That buffer can be mapped to LM, SM (snooped or unsnooped) and can be linear or tiled. When both the Depth and Color buffers are tiled, the respective Tile Walk directions must match.

When a linear Color and a linear Depth buffers are used together:

1. They may have different pitches, though both pitches must be a multiple of 32 bytes.

2. They must be co-aligned with a 32-byte region.

7.16 3D Depth Buffer Surfaces

Depth buffer surfaces are used to hold per-pixel depth values and per-pixel stencil values for use in the 3D pipeline. Note that the 3D pipeline does not require a Depth buffer to be allocated, though a Depth buffer is required to perform (non-trivial) Depth Test and Stencil Test operations.

The following table summarizes the possible formats of the Depth buffer. Refer to Depth Buffer Formats section in this chapter for details on the pixel formats. Refer to the *Windower* and *DataPort* chapters for details on the usage of the Depth Buffer.

Table 7-22. Depth Buffer Formats

DepthBufferFormat / DepthComponent	bpp	Description
D32_FLOAT_S8X24_UINT	64	32-bit floating point Z depth value in first DWord, 8-bit stencil in lower byte of second DWord
D32_FLOAT	32	32-bit floating point Z depth value
D24_UNORM_S8_UINT	32	24-bit fixed point Z depth value in lower 3 bytes, 8-bit stencil value in upper byte
D16_UNORM	16	16-bit fixed point Z depth value

The Depth buffer is specified via the 3DSTATE_DEPTH_BUFFER command. See the description of that instruction in *Windower* for restrictions.

7.17 3D Separate Stencil Buffer Surfaces [DevILK+]

Separate Stencil buffer surfaces are used to hold per-pixel stencil values for use in the 3D pipeline. Note that the 3D pipeline does not require a Stencil buffer to be allocated, though a Stencil buffer is required to perform (non-trivial) Stencil Test operations.

The following table summarizes the possible formats of the Stencil buffer. Refer to Stencil Buffer Formats section in this chapter for details on the pixel formats. Refer to the *Windower* chapters for details on the usage of the Stencil Buffer.

Table 7-23. Depth Buffer Formats

DepthBufferFormat / DepthComponent	bpp	Description
S8_UINT	8	8-bit stencil value in a byte

The Stencil buffer is specified via the 3DSTATE_STENCIL_BUFFER command. See the description of that instruction in *Windower* for restrictions.

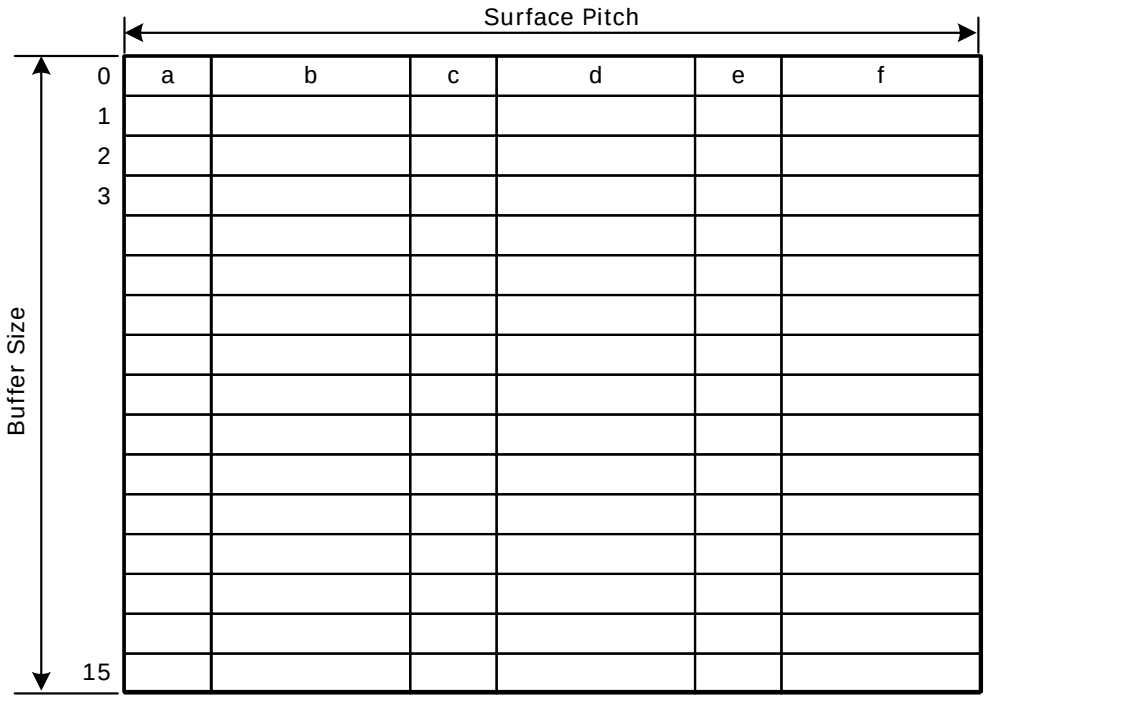
7.18 Surface Layout

This section describes the formats of surfaces and data within the surfaces.

7.18.1 Buffers

A buffer is an array of structures. Each structure contains up to 2048 bytes of elements. Each element is a single surface format using one of the supported surface formats depending on how the surface is being accessed. The surface pitch state for the surface specifies the size of each structure in bytes.

The buffer is stored in memory contiguously with each element in the structure packed together, and the first element in the next structure immediately following the last element of the previous structure. Buffers are supported only in linear memory.



7.18.2 1D Surfaces

One-dimensional surfaces are identical to 2D surfaces with height of one. Arrays of 1D surfaces are also supported. Please refer to the 2D Surfaces section for details on how these surfaces are stored.

7.18.3 2D Surfaces

Surfaces that comprise texture mip-maps are stored in a fixed "monolithic" format and referenced by a single base address. The base map and associated mipmaps are located within a single rectangular area of memory identified by the base address of the upper left corner and a pitch. The base address references the upper left corner of the base map. The pitch must be specified at least as large as the widest mip-map. In some cases it must be wider; see the section on Minimum Pitch below.

These surfaces may be overlapped in memory and must adhere to the following memory organization rules:

- For non-compressed texture formats, each mipmap must start on an even row within the monolithic rectangular area. For 1-texel-high mipmaps, this may require a row of padding below the previous mipmap. This restriction does not apply to any compressed texture formats: i.e., each subsequent (lower-res) compressed mipmap is positioned directly below the previous mipmap.

- Vertical alignment restrictions vary with memory tiling type: 1 DWord for linear, 16-byte (DQWord) for tiled. (Note that tiled mipmaps are *not* required to start at the left edge of a tile row).

7.18.3.1 Computing MIP level sizes

Map width and height specify the size of the largest MIP level (LOD 0). Less detailed LOD level (i+1) sizes are determined by dividing the width and height of the current (i) LOD level by 2 and truncating to an integer (floor). This is equivalent to shifting the width/height by 1 bit to the right and discarding the bit shifted off. The map height and width are clamped on the low side at 1.

In equations, the width and height of an LOD "*L*" can be expressed as:

$$W_L = ((width >> L) > 0 ? width >> L : 1)$$
$$H_L = ((height >> L) > 0 ? height >> L : 1)$$

[DevSNB+]: If the surface is multisampled (4x), these values must be adjusted as follows before proceeding:

$$W_L = ceiling(W_L / 2) * 4$$
$$H_L = ceiling(H_L / 2) * 4$$

7.18.3.2 Base Address for LOD Calculation

It is conceptually easier to think of the space that the map uses in Cartesian space (x, y), where x and y are in units of texels, with the upper left corner of the base map at (0, 0). The final step is to convert from Cartesian coordinates to linear addresses as documented at the bottom of this section.

It is useful to think of the concept of "stepping" when considering where the next MIP level will be stored in rectangular memory space. We either step down or step right when moving to the next higher LOD.

- for MIPLAYOUT_RIGHT maps:
  - step right when moving from LOD 0 to LOD 1
  - step down for all of the other MIPs
- for MIPLAYOUT_BELOW maps:
  - step down when moving from LOD 0 to LOD 1
  - step right when moving from LOD 1 to LOD 2
  - step down for all of the other MIPs

To account for the cache line alignment required, we define *i* and *j* as the width and height, respectively, of an *alignment unit*. This alignment unit is defined below. We then define lower-case w_L and h_L as the padded width and height of LOD "*L*" as follows:

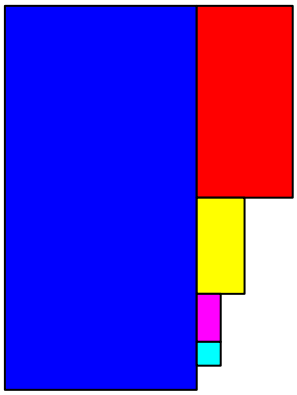
$$w_L = i * ceil\left(\frac{W_L}{i}\right)$$

$$h_L = j * ceil\left(\frac{H_L}{j}\right)$$

Equations to compute the upper left corner of each MIP level are then as follows:

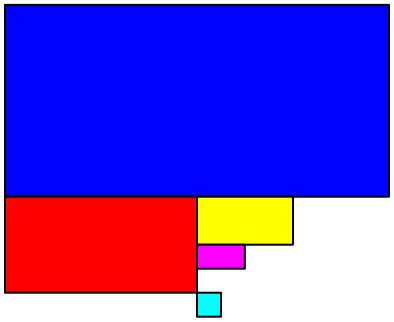
for *MIPLAYOUT_RIGHT* maps:

$$LOD_0 = (0,0)$$
$$LOD_1 = (w_0, 0)$$
$$LOD_2 = (w_0, h_1)$$
$$LOD_3 = (w_0, h_1 + h_2)$$
$$LOD_4 = (w_0, h_1 + h_2 + h_3)$$
$$\ldots$$



for *MIPLAYOUT_BELOW* maps:

$$LOD_0 = (0,0)$$
$$LOD_1 = (0, h_0)$$
$$LOD_2 = (w_1, h_0)$$
$$LOD_3 = (w_1, h_0 + h_2)$$
$$LOD_4 = (w_1, h_0 + h_2 + h_3)$$
$$\ldots$$



7.18.3.3 Minimum Pitch

For MIPLAYOUT_RIGHT maps, the minimum pitch must be calculated before choosing a fence to place the map within. This is approximately equal to 1.5x the pitch required by the base map, with possible adjustments made for cache line alignment. For MIPLAYOUT_BELOW and MIPLAYOUT_LEGACY maps, the minimum pitch required is equal to that required by the base (LOD 0) map.

A safe but simple calculation of minimum pitch is equal to 2x the pitch required by the base map for MIPLAYOUT_RIGHT maps. This ensures that enough pitch is available, and since it is restricted to MIPLAYOUT_RIGHT maps, not much memory is wasted. It is up to the driver (hardware independent) whether to use this simple determination of pitch or a more complex one.

7.18.3.4 Alignment Unit Size

The following table indicates the *i* and *j* values that should be used for each map format. Note that the compressed formats are padded to a full compression cell.

Table 7-24. Alignment Units for Texture Maps

surface format	alignment unit width "*i*"	alignment unit height "*j*"
YUV 4:2:2 formats	4	* see below
BC1-5	4	4
FXT1	8	4
all other formats	4	* see below

* For these formats, the vertical alignment factor "j" is determined as follows:

- For [DevSNB+]:
  - j = 4 for any depth buffer
  - j = 2 for separate stencil buffer
  - j = 4 for any render target surface is multisampled (4x)
  - j = 4 for any render target surface with **Surface Vertical Alignment** = VALIGN_4
  - j = 2 for any render target surface with **Surface Vertical Alignment** = VALIGN_2
  - j = 2 for all other render target surface
  - j = 2 for any sampling engine surface with **Surface Vertical Alignment** = VALIGN_2
  - j = 4 for any sampling engine surface with **Surface Vertical Alignment** = VALIGN_4

7.18.3.5 Cartesian to Linear Address Conversion

A set of variables are defined in addition to the i and j defined above.

- b = bytes per texel of the native map format (0.5 for DXT1, FXT1, and 4-bit surface format, 2.0 for YUV 4:2:2, others aligned to surface format)
- t = texel rows / memory row (4 for DXT1-5 and FXT1, 1 for all other formats)
- p = pitch in bytes (equal to pitch in dwords * 4)
- B = base address in bytes (address of texel 0,0 of the base map)
- x, y = cartestian coordinates from the above calculations in units of texels (assumed that x is always a multiple of i and y is a multiple of j)
- A = linear address in bytes

$$A = B + \frac{yp}{t} + xbt$$

This calculation gives the linear address in bytes for a given MIP level (taking into account L1 cache line alignment requirements).

7.18.3.6 Compressed Mipmap Layout

Mipmaps of textures using compressed (DXTn, FXT) texel formats are also stored in a monolithic format. The compressed mipmaps are stored in a similar fashion to uncompressed mipmaps, with each block of source (uncompressed) texels represented by a 1 or 2 QWord compressed block. The compressed blocks occupy the same logical positions as the texels they represent, where each row of compressed blocks represent a 4-high row of uncompressed texels. The format of the blocks is preserved, i.e., there is no “intermediate” format as required on some other devices.

The following exceptions apply to the layout of compressed (vs. uncompressed) mipmaps:

- Mipmaps are not required to start on even rows, therefore each successive mip level is located on the texel row immediately below the last row of the previous mip level. Pad rows are neither required nor allowed.
- The dimensions of the mip maps are first determined by applying the sizing algorithm presented in Non-Power-of-Two Mipmaps above. Then, if necessary, they are padded out to compression block boundaries.

7.18.3.7 Surface Arrays

7.18.3.7.1 For all surfaces other than separate stencil buffer

Both 1D and 2D surfaces can be specified as an array. The only difference in the surface state is the presence of a depth value greater than one, indicating multiple array “slices”.

A value *QPitch* is defined which indicates the worst-case height for one slice in the texture array. This *QPitch* is multiplied by the array index to and added to the vertical component of the address to determine the vertical component of the address for that slice. Within the slice, the map is stored identically to a MIPLAYOUT_BELOW 2D surface. *MIPLAYOUT_BELOW is the only format supported by 1D non-arrays*

and both 2D and 1D arrays, the programming of the MIP Map Layout Mode state variable is ignored when using a TextureArray.

The following equation is used for surface formats other than compressed textures:

$$QPitch = (h_0 + h_1 + 11j)$$

The input variables in this equation are defined in sections above.

The equation for compressed textures (BC* and FXT1 surface formats) follows:

$$QPitch = \frac{(h_0 + h_1 + 11j)}{4}$$

[DevSNB] Errata: Sampler MSAA Qpitch will be 4 greater than the value calculated in the equation above , for every other odd Surface Height starting from 1 i.e. 1,5,9,13

7.18.3.7.2 For separate stencil buffer [DevILK] to [DevSNB]

The separate stencil buffer does not support mip mapping, thus the storage for LODs other than LOD 0 is not needed. The following *QPitch* equation applies only to the separate stencil buffer:

$$QPitch = h_0$$

7.18.4 Cube Surfaces

The 3D pipeline supports *cubic environment maps*, conceptually arranged as a cube surrounding the origin of a 3D coordinate system aligned to the cube faces. These maps can be used to supply texel (color/alpha) data of the environment in any direction from the enclosed origin, where the direction is supplied as a 3D "vector" texture coordinate. These cube maps can also be mipmapped.

Each texture map level is represented as a group of six, square *cube face* texture surfaces. The faces are identified by their relationship to the 3D texture coordinate system. The subsections below describe the cube maps as described at the API as well as the memory layout dictated by the hardware.

7.18.4.1 Hardware Cube Map Layout

7.18.4.1.1 Hardware Cube Map layout [DevILK+]

The cube face textures are stored in the same way as 2D array surfaces are stored (see section 7.18.3 for details). For cube surfaces, the depth (array instances) is equal to 6. The array index "q" corresponds to the face according to the following table:

"q" coordinate	face
0	+x

1	-x
2	+y
3	-y
4	+z
5	-z

7.18.4.2 Restrictions

- The cube map memory layout is the same whether or not the cube map is mip-mapped, and whether or not all six faces are “enabled”, though the memory backing disabled faces or non-supplied levels can be used by software for other purposes.

- The cube map faces all share the same **Surface Format.**

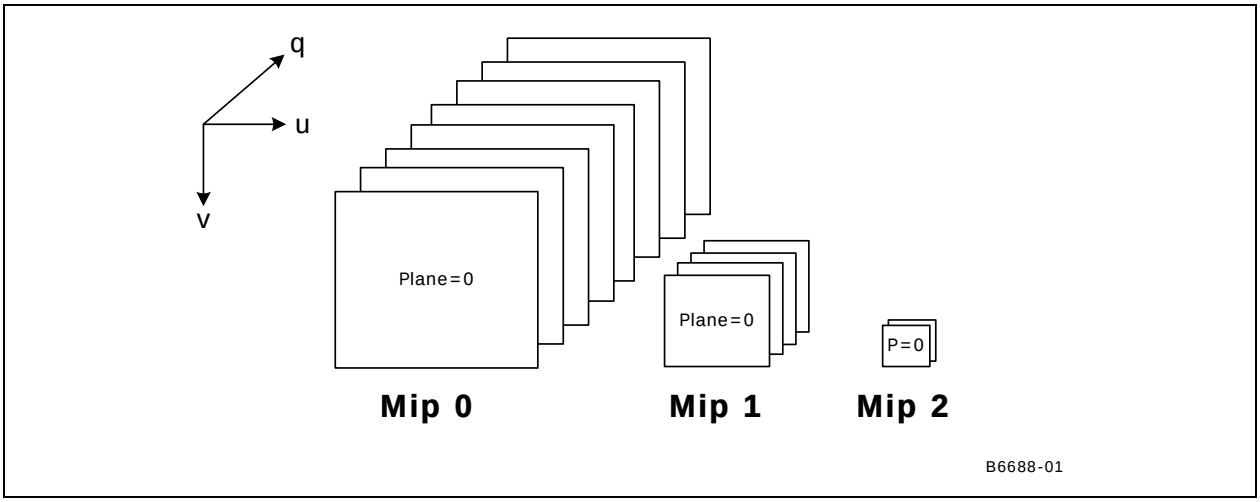
7.18.4.3 Cube Arrays [DevSNB+]

Cube arrays are stored identically to 2D surface arrays. A group of 6 consecutive array elements makes up a single cube map. A cube array with N array elements is stored identically to a 2D array with 6N array elements.

7.18.5 3D Surfaces

Multiple texture map surfaces (and their respective mipmap chains) can be arranged into a structure known as a Texture3D (volume) texture. A volume texture map consists of many *planes* of 2D texture maps. See *Sampler* for a description of how volume textures are used.

Figure 7-5. Volume Texture Map



Note that the number of planes defined at each successive mip level is halved. Volumetric texture maps are stored as follows. All of the LOD=0 q-planes are stacked vertically, then below that, the LOD=1 q-planes are stacked two-wide, then the LOD=2 q-planes are stacked four-wide below that, and so on.

The width, height, and depth of LOD "L" are as follows:

$$W_L = ((width >> L) > 0 ? width >> L : 1)$$
$$H_L = ((height >> L) > 0 ? height >> L : 1)$$

This is the same as for a regular texture. For volume textures we add:

$$D_L = ((depth >> L) > 0 ? depth >> L : 1)$$

Cache-line aligned width and height are as follows, with i and j being a function of the map format as shown in the *Alignment Unit Size* section.

$$w_L = i * ceil\left(\frac{W_L}{i}\right)$$

$$h_L = j * ceil\left(\frac{H_L}{j}\right)$$

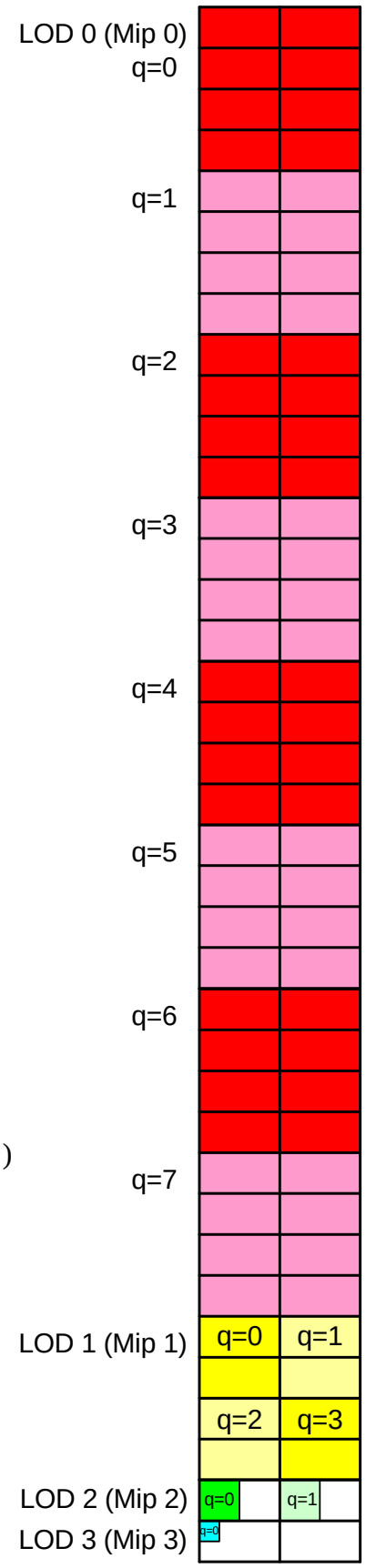
Note that it is not necessary to cache-line align in the "depth" dimension (i.e. lower case "*d*").

The following equations for $LOD_{L,q}$ give the base address Cartesian coordinates for the map at LOD L and depth q.

$$LOD_{0,q} = (0, q * h_0)$$
$$LOD_{1,q} = ((q\%2) * w_1, D_0 * h_0 + (q >> 1) * h_1)$$
$$LOD_{2,q} = ((q\%4) * w_2, D_0 * h_0 + ceil\left(\frac{D_1}{2}\right) * h_1 + (q >> 2) * h_2)$$
$$LOD_{3,q} = ((q\%8) * w_3, D_0 * h_0 + ceil\left(\frac{D_1}{2}\right) * h_1 + ceil\left(\frac{D_2}{4}\right) * h_2 + (q >> 3) * h_3)$$

...

These values are then used as "base addresses" and the 2D MIP Map equations are used to compute the location within each LOD/q map.



7.18.5.1 Minimum Pitch

The minimum pitch required to store the 3D map may in some cases be greater than the minimum pitch required by the LOD=0 map. This is due to

cache line alignment requirements that may impact some of the MIP levels requiring additional spacing in the horizontal direction.

7.19 Surface Padding Requirements

7.19.1 Sampling Engine Surfaces

The sampling engine accesses texels outside of the surface if they are contained in the same cache line as texels that are within the surface. These texels will not participate in any calculation performed by the sampling engine and will not affect the result of any sampling engine operation, however if these texels lie outside of defined pages in the GTT, a GTT error will result when the cache line is accessed. In order to avoid these GTT errors, “padding” at the bottom and right side of a sampling engine surface is sometimes necessary.

It is possible that a cache line will straddle a page boundary if the base address or pitch is not aligned. All pages included in the cache lines that are part of the surface must map to valid GTT entries to avoid errors. To determine the necessary padding on the bottom and right side of the surface, refer to the table in Section 7.18.3.4 for the i and j parameters for the surface format in use. The surface must then be extended to the next multiple of the alignment unit size in each dimension, and all texels contained in this extended surface must have valid GTT entries.

For example, suppose the surface size is 15 texels by 10 texels and the alignment parameters are i=4 and j=2. In this case, the extended surface would be 16 by 10. Note that these calculations are done in texels, and must be converted to bytes based on the surface format being used to determine whether additional pages need to be defined.

For buffers, which have no inherent “height,” padding requirements are different. A buffer must be padded to the next multiple of 256 array elements, with an additional 16 bytes added beyond that to account for the L1 cache line.

For cube surfaces, an additional two rows of padding are required at the bottom of the surface. This must be ensured regardless of whether the surface is stored tiled or linear. This is due to the potential rotation of cache line orientation from memory to cache.

For compressed textures (BC* and FXT1 surface formats), padding at the bottom of the surface is to an even compressed row, which is equal to a multiple of 8 uncompressed texel rows. Thus, for padding purposes, these surfaces behave as if j = 8 only for surface padding purposes. The value of 4 for j still applies for mip level alignment and QPitch calculation.

For YUV, 96 bpt, and 48 bpt surface formats, additional padding is required. These surfaces require an extra row plus 16 bytes of padding at the bottom in addition to the general padding requirements.

7.19.2 Render Target and Media Surfaces

The data port accesses data (pixels) outside of the surface if they are contained in the same cache request as pixels that are within the surface. These pixels will not be returned by the requesting message, however if these pixels lie outside of defined pages in the GTT, a GTT error will result when the cache request is processed. In order to avoid these GTT errors, “padding” at the bottom of the surface is sometimes necessary.

If the surface contains an odd number of rows of data, a final row below the surface must be allocated. If the surface will be accessed in field mode (**Vertical Stride** = 1), enough additional rows below the surface must be allocated to make the extended surface height (including the padding) a multiple of 4.

7.19.3 Register/State Context [DevSNB+]

		Valid Only When PPGTT Enabled					
DW Range	**DW Count**	**State Field**	**Restore Inhibited**	**PPGTT Enabled**	**PPGTT Disabled**	**Power Context**	**Set Before Submitting Context?**
00h	1	Context Control	R	S/R	X	S/R	Yes
01h	1	Ring Head Pointer Register	R	S/R	X	S/R	Yes
02h	1	Ring Tail Pointer Register	R	R	X	S/R	Yes
03h	1	Batch Buffer Current Head Register	NR	S/R	X	S/R	No
04h	1	Batch Buffer State Register	NR	S/R	X	S/R	No
05h	1	PPGTT Directory Cache Valid Register (Software always populates via host)	R	R	X	S/R	Yes
06h	1	**Reserved**	X	X	X	S/R	X
07h	1	PD Base Virtual Address Register	R	R	X	S/R	Yes
08h	1	MFX_STATE_POINTER 0	NR	S/R	X	S/R	Yes
09h	1	MFX_STATE_POINTER 1	NR	S/R	X	S/R	Yes
0Ah	1	MFX_STATE_POINTER 2	NR	S/R	X	S/R	Yes
0Bh	1	MFX_STATE_POINTER 3	NR	S/R	X	S/R	Yes
0Ch	1	VCS_CNTR— Media Watchdog Counter Control	NR	S/R	X	S/R	No
0Dh	1	VCS_THRSH— Media Watchdog Counter Threshold	NR	S/R	X	S/R	No
0Eh	1	Current Context ID Register	NR	S/R	X	S/R	No
0Fh	1	**Reserved**	X	X	X	S/R	X

7.19.4 The Per-Process Hardware Status Page

The following table defines the layout of the Per-process Hardware Status Page:

DWord Offset	Description
(3FFh – 020h)	These locations can be used for general purpose via the MI_STORE_DATA_INDEX or MI_STORE_DATA_IMM instructions.
1F:5	**Reserved.**
4	**Ring Head Pointer Storage:** The contents of the Ring Buffer Head Pointer register (register DWord 1) are written to this location either as result of an MI_REPORT_HEAD instruction or as the result of an "automatic report" (see RINGBUF registers).
3:0	**Reserved.**

This page is designed to be read by SW in order to glean additional details about a context beyond what it can get from the context status.

Accesses to this page will automatically be treated as cacheable and snooped. It is therefore illegal to locate this page in any region where snooping is illegal (such as in stolen memory).

7.19.5 Register/State Context

		Valid Only When PPGTT Enabled					
DW Range	**DW Count**	**State Field**	**Render Restore Inhibited**	**PPGTT Enabled**	**PPGTT Disabled**	**Power Context**	**Set Before Submitting Context?**
00h	1	**Reserved**	NR	X	X	X	X
01h	1	Ring Head Pointer Register	R	S/R	X	S/R	Yes
02h	1	Ring Tail Pointer Register	R	R	X	S/R	Yes
03h	1	**Reserved**	NR	X	X	X	X
04h	1	**Reserved**	NR	X	X	X	X
05h	1	PPGTT Directory Cache Valid Register	R	R	X	X	Yes

		Valid Only When PPGTT Enabled					
DW Range	**DW Count**	**State Field**	**Render Restore Inhibited**	**PPGTT Enabled**	**PPGTT Disabled**	**Power Context**	**Set Before Submitting Context?**
		(Software always populates via host)					
06h	1	BCS_SWCTRL Register	NR	S/R	X	S/R	Yes
07h	1	PD Base Virtual Address Register	R	R	X	X	Yes
08h	1	**Reserved**	NR	X	X	X	X
09h	1	**Reserved**	NR	X	X	X	X
0Ah	1	**Reserved**	NR	X	X	X	X
0Bh	1	**Reserved**	NR	X	X	X	X
0Ch	1	**Reserved**	NR	X	X	X	X
0Dh	1	**Reserved**	NR	X	X	X	X
0Eh	1	**Reserved**	NR	X	X	X	X
0Fh	1	**Reserved**	NR	X	X	X	X

7.19.6 The Per-Process Hardware Status Page

The following table defines the layout of the Per-process Hardware Status Page:

DWord Offset	Description
(3FFh – 020h)	These locations can be used for general purpose via the MI_STORE_DATA_INDEX or MI_STORE_DATA_IMM instructions.
1F:5	**Reserved.**
4	**Ring Head Pointer Storage:** The contents of the Ring Buffer Head Pointer register (register DWord 1) are written to this location either as result of an MI_REPORT_HEAD instruction or as the result of an “automatic report” (see RINGBUF registers).
3:0	**Reserved.**

This page is designed to be read by SW in order to glean additional details about a context beyond what it can get from the context status.

Accesses to this page will automatically be treated as cacheable and snooped. It is therefore illegal to locate this page in any region where snooping is illegal (such as in stolen memory).

7.19.7 Overall Context Layout

7.19.7.1 Per-Process GTT Disabled

For this case, the entire context image consists of the *Register/State Context*, including the pipelined state section.

7.19.7.2 Per-Process GTT Enabled [DevSNB+]

For [DevSNB+], when PPGTT is enabled, the Context Image for the rendering engine consists of 11 4K pages:

Register/State Context
Probe List
Pipelined State (4 pages, 16KB)
Ring Buffer (4 Pages, 16KB)
Per-Process HW Status Page

The pipelined state is not saved as part of the *Register/State Context* in this mode, but instead goes into a separate page.

7.19.8 Register/State Context

7.19.8.1 Register/State Context [DevSNB]

The *Register/State Context* breaks down into cachelines as follows:

CL #	Description
0h	Ring Registers and AS-Specific Pipe Context Data (AS Only)
	Contains the only DWs required to be initialized in the image by SW
1h-2h	Probe Valid Registers (AS Only)
3h-Ah	Non-Pipelined 3D State Context Data
Bh-Dh	Media Context
Eh	Media PRT
10h-30h	Sampler Palette Load (Extended Only)
31h-33h	Poly Stipple Pattern (Extended Only)
34h-39h	Image Enhancement
0h	Pipelined 3D (Stored Here Only When PPGTT Disabled)
13h-1Fh	Reserved

Ring Registers and Non-Pipelined Context Details:

		Valid Only When PPGTT Enabled					
DW Range	**DW Count**	**State Field**	**Render Restore Inhibited**	**PPGTT Enabled**	**PPGTT Disabled**	**Power Context**	**Set Before Submitting Context?**
00h	1	Context Control	R	S/R	X	X	Yes
01h	1	Ring Head Pointer Register	R	S/R	X	S/R	Yes
02h	1	Ring Tail Pointer Register	R	R	X	S/R	Yes
03h	1	Batch Buffer Current Head Register	NR	S/R	X	X	No
04h	1	Batch Buffer State Register [DevSNB] NR	NR	S/R	X	X [DevSNB] NR	No
05h	1	PPGTT Directory Cache Valid Register	R	R	X	X	Yes
06h	1	**Reserved** (for PPGTT Directory Cache Valid High)	NR	X	X	X	X

		Valid Only When PPGTT Enabled					
DW Range	**DW Count**	**State Field**	**Render Restore Inhibited**	**PPGTT Enabled**	**PPGTT Disabled**	**Power Context**	**Set Before Submitting Context?**
07h	1	PD Base Virtual Address Register	R	R	X	X	Yes
08h	1	Read Offset in Piipelined State Page and PAVP flags ((8 CL aligned)	NR	S/R	X	X	No
09h	1	Committed Vertex Number	NR	S/R	X	X	No
0Ah	1	Committed Instance ID	NR	S/R	X	X	No
0Bh	1	Committed Primitive ID	NR	S/R	X	X	No
0Ch	1	Super Span Count	NR	S/R	X	X	No
0Dh	1	VDI Walker Data	NR	S/R	X	X	No
0Eh	1	CCID Register	NR	X	X	X	X
0Fh	1	**Reserved**	NR	X	X	X	X
10h – 1Fh	16	Probe Valid Registers	R	S/R	X	X	Yes
20h – 2Fh	16	Probe Valid Registers	R	S/R	X	X	Yes
30h – 31h	2	IA_VERTICES_COUNT Register	NR	S/R	S/R	S/R	No
32h – 33h	2	IA_PRIMITIVES_COUNT Register	|	|	|	|	|
34h – 35h	2	VS_INVOCATION_COUNT Register	V	V	V	V	V
36h – 37h	2	GS_INVOCATION_COUNT Register					
38h – 39h	2	Num Primitives Written Register					
3Ah – 3Bh	2	Primitive Storage Needed Register					
3Ch	1	Streaming Vertex Buffer Index 0					
3Dh	1	Streaming Vertex Buffer Index 1					
3Eh	1	Streaming Vertex Buffer Index 2					
3Fh	1	Streaming Vertex Buffer Index 3					
40h – 41h	2	GS_PRIMITIVES_COUNT Register					

		Valid Only When PPGTT Enabled					
DW Range	**DW Count**	**State Field**	**Render Restore Inhibited**	**PPGTT Enabled**	**PPGTT Disabled**	**Power Context**	**Set Before Submitting Context?**
42h –43h	2	CL_INVOCATION_COUNT Register					
44h – 45h	2	CL_PRIMITIVES_COUNT Register					
46h – 47h	2	PS_INVOCATION_COUNT Register					
48h – 49h	2	PS_DEPTH_COUNT Register					
4Ah	1	CACHE_MODE_0 Register					
4Bh	1	CACHE_MODE_1 Register					
4Ch	1	**Reserved**					
4Dh	1	INSTPM Register					
4Eh	1	EXCC Register					
4Fh	1	MI_MODE Register					
50h	1	Max Streaming Vertex Buffer Index 0					
51h	1	Max Streaming Vertex Buffer Index 1					
52h	1	Max Streaming Vertex Buffer Index 2					
53h	1	Max Streaming Vertex Buffer Index 3					
54h	1	Render Watchdog Counter Control					
55h	1	Render Watchdog Counter Threshold					
56h	1	FBC RC Base Address					
57h	1	**Reserved**					
58h	1	RVSYNC Register					
59h	1	RBSYNC Register					
5Ah – 5Bh	2	GW Timestamp Delta Value					
5Ch	1	TIMESTAMP Register (LSB)					

		Valid Only When PPGTT Enabled					
DW Range	**DW Count**	**State Field**	**Render Restore Inhibited**	**PPGTT Enabled**	**PPGTT Disabled**	**Power Context**	**Set Before Submitting Context?**
5Dh	1	VFE Debug Counter					
5E	1	ARB_OFF_CTR Register					
5Fh	1	ARB_OFF_THRSH Register					
60h	1	PIPELINE_SELECT					
61h – 6Ah	10	STATE_BASE_ADDRESS					
6Bh – 6Fh	5	**Reserved**					
70h – 71h	2	STATE_SIP					
72h – 75h	4	3DSTATE_DRAWING_RECTANGLE					
76h – 78h	3	3DSTATE_AA_LINE_PARAMS					
79h – 7Fh	7	3DSTATE_DEPTH_BUFFER					
80h – 84h	5	**Reserved**					
85h – 86h	2	3DSTATE_POLY_STIPPLE_OFFSET					
87h – 89h	3	3DSTATE_LINE_STIPPLE					
8Ah – 8Fh	6	**Reserved**					
90h – 92h	3	3DSTATE_HIER_DEPTH_BUFFER					
93h – 95h	3	3DSTATE_STENCIL_BUFFER					
96h – 97h	2	3DSTATE_CLEAR_PARAMS					
98h – 99h	2	3DSTATE_MONOFILTER_SIZE					
9Ah – 9Ch	3	3DSTATE_MULTISAMPLE					
9Dh – 9Fh	3	**Reserved**					
A0h – A3h	4	3DSTATE_CHROMA_KEY (0)					
A4h – A7h	4	3DSTATE_CHROMA_KEY (1)					

		Valid Only When PPGTT Enabled					
DW Range	**DW Count**	**State Field**	**Render Restore Inhibited**	**PPGTT Enabled**	**PPGTT Disabled**	**Power Context**	**Set Before Submitting Context?**
A8h – ABh	4	3DSTATE_CHROMA_KEY (2)					
ACh – AFh	4	3DSTATE_CHROMA_KEY (3)					
B0h – B7h	8	MEDIA_VFE_STATE					
B8h – BBh	4	MEDIA_CURBE_LOAD					
BCh – BFh	4	MEDIA_INTERFACE_DESCRIPTOR_LOAD					
C0h - DFh	32	GATEWAY_BARRIER					
E0h – EFh	16	Media Object PRT Data					
F0h – FFh	16	**Reserved**					
100h – 200h	257	3DSTATE_SAMPLER_PALETTE_LOAD_0					
201h – 206h	6	**Reserved**					
207h – 307h	257	3DSTATE_SAMPLER_PALETTE_LOAD_1					
308h – 30Fh	8	**Reserved**					
310h – 330h	33	3DSTATE_POLY_STIPPLE_PATTERN					
331h – 33Fh	15	**Reserved**					
340h – 34Fh	16	FMD Registers (MMIO 5000h-503Fh)					
350h – 359h	10	FMD Registers (MMIO 5040h-5067h)					
35Ah – 35Bh	2	STD Ymin/Ymax/#skin pixels (MMIO 5070h-5077h)					
35Ch – 35Fh	4	**Reserved**					
350h – 39Fh	64	ACE Histogram Registers (bins 0–127) (MMIO 5080h-517Fh)					

		Valid Only When PPGTT Enabled					
DW Range	**DW Count**	**State Field**	**Render Restore Inhibited**	**PPGTT Enabled**	**PPGTT Disabled**	**Power Context**	**Set Before Submitting Context?**
0h – 2h	3	3DSTATE_INDEX_BUFFER					
3h – 87h	133	3DSTATE_VERTEX_BUFFERS					
88h – CCh	69	3DSTATE_VERTEX_ELEMENTS					
CDh	1	3DSTATE_VF_STATISTICS					
CEh – CFh	2	**Reserved**					
D0h – D3h	4	3DSTATE_BINDING_TABLE_POINTERS					
D4h – D6h	3	3DSTATE_URB					
D7h – DAh	4	3DSTATE_CC_STATE_POINTERS					
DBh – DEh	4	3DSTATE_SAMPLER_STATE_POINTERS					
DFh – E2h	4	3DSTATE_VIEWPORT_STATE_POINTERS					
E3h – E4h	2	3DSTATE_SCISSOR_STATE_POINTERS					
E5h – EAh	6	3DSTATE_VS					
EBh – F1h	7	3DSTATE_GS					
F2h – F5h	4	3DSTATE_CLIP					
F6h – 109h	20	3DSTATE_SF					
10Ah – 112h	9	3DSTATE_WM					
113h – 114h	2	3DSTATE_SAMPLE_MASK					
115h – 119h	5	3DSTATE_CONSTANT_VS					
11Ah – 11Eh	5	3DSTATE_CONSTANT_GS					
11Fh – 123h	5	3DSTATE_CONSTANT_PS					
124h – 1FFh	220	**Reserved**					

7.19.9 Pipelined State Page

This page is used a scratch area for the pipeline to store pipelined state that is not referenced indirectly. Under no circumstances should SW read from or write to this page.

7.19.10 Ring Buffer

This page is used a scratch area for the pipeline to store ring buffer commands that need to be reissued. Under no circumstances should SW read from or write to this page.

7.19.11 The Per-Process Hardware Status Page

The following table defines the layout of the Per-process Hardware Status Page:

DWord Offset	Description
(3FFh – 020h)	These locations can be used for general purpose via the MI_STORE_DATA_INDEX or MI_STORE_DATA_IMM instructions.
1F:1C	**Reserved.**
1B	**Context Save Finished Timestamp**
1A	**Context Restore Complete Timestamp**
19	**Pre-empt Request Received Timestamp**
18	**Last Switch Timestamp**
17:12	**Reserved.**
11:10	**Probe List Slot Fault Register (2 DWs)**
F:5	**Reserved.**
4	**Ring Head Pointer Storage:** The contents of the Ring Buffer Head Pointer register (register DWord 1) are written to this location either as result of an MI_REPORT_HEAD instruction or as the result of an “automatic report” (see RINGBUF registers).
3:0	**Reserved.**

This page is designed to be read by SW in order to glean additional details about a context beyond what it can get from the context status.

Accesses to this page will automatically be treated as cacheable and snooped. It is therefore illegal to locate this page in any region where snooping is illegal (such as in stolen memory).

Revision History

Revision Number	Description	Revision Date
1.0	First 2011 OpenSource edition	May 2011

§§

Intel® OpenSource HD Graphics Programmer's Reference Manual (PRM)

Volume 1 Part 2: Graphics Core – MMIO, Media Registers, and Programming Environment (SandyBridge)

For the 2011 Intel Core Processor Family

May 2011

Revision 1.0

NOTICE:

This document contains information on products in the design phase of development, and Intel reserves the right to add or remove product features at any time, with or without changes to this open source documentation.

Doc Ref #: IHD-OS-V1 Pt2 – 05 11

Creative Commons License

You are free: to Share — to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

No Derivative Works. You may not alter, transform, or build upon this work.

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

Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The SandyBridge chipset family, Havendale/Auburndale chipset family, Intel® 965 Express Chipset Family, Intel® G35 Express Chipset, and Intel® 965GMx Chipset Mobile Family Graphics Controller may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order. I2C is a two-wire communications bus/protocol developed by Philips. SMBus is a subset of the I2C bus/protocol and was developed by Intel.

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

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

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

Copyright © 2011, Intel Corporation. All rights reserved.

Contents

1. Introduction ... 5

2. Virtual Memory ... 6

2.1 Virtual Memory Control ... 6
2.2 Global Virtual Memory ... 6
2.2.1 Graphics Translation Table (GTT) Range (GTTADR) ... 7
2.2.2 GTT Page Table Entries (PTEs) ... 7
2.3 Two-Level Per-Process Virtual Memory ... 8
2.3.1 PPGTT Directory Entries (PDEs) ... 10
2.3.2 PPGTT Table Entries (PTEs) ... 10
2.3.3 Context TLBs in DevSNB A-step ... 11
2.3.4 PPGTT vs GGTT limitation ... 11

3. GFX MMIO – MCHBAR Aperture ... 12

4. Graphics Memory Interface Functions ... 13

4.1 Introduction ... 13
4.2 Graphics Memory Clients ... 13
4.3 Graphics Memory Addressing Overview ... 14
4.3.1 Graphics Address Path ... 14
4.4 Graphics Memory Address Spaces ... 15
4.5 Address Tiling Function ... 16
4.5.1 Linear vs. Tiled Storage ... 16
4.5.2 Tile Formats ... 20
4.5.3 Tiling Algorithm ... 24
4.5.4 Tiling Support ... 26
4.5.5 Per-Stream Tile Format Support ... 32
4.6 Logical Memory Mapping ... 33
4.6.1 Changes to GTT ... 33
4.6.2 Logical Memory Space Mappings ... 34
4.7 Physical Graphics Memory ... 38
4.7.1 Physical Graphics Address Types ... 38
4.7.2 Main Memory ... 39

5. Device Programming Environment ... 41

5.1 Programming Model ... 41
5.2 Graphics Device Register Programming ... 41
5.3 Graphics Device Command Streams ... 42
5.3.1 Command Use ... 42
5.3.2 Command Transport Overview ... 42
5.3.3 Command Parser ... 42
5.3.4 The Ring Buffer ... 43
5.3.5 Batch Buffers ... 47
5.3.6 Indirect Data ... 48
5.3.7 Command Arbitration ... 48
5.3.8 Graphics Engine Synchronization ... 51
5.3.9 Graphics Memory Coherency ... 52
5.3.10 Graphics Cache Coherency ... 52
5.3.11 Command Synchronization ... 54
5.4 Hardware Status ... 56
5.4.1 Hardware-Detected Errors (Master Error bit) ... 57
5.4.2 Thermal Sensor Event ... 57
5.4.3 Sync Status ... 57

5.4.4 Display Plane A, B, (Sprite A, Sprite B) Flip Pending 57
5.4.5 Display Pipe A,B VBLANK 57
5.4.6 User Interrupt 57
5.4.7 PIPE_CONTROL Notify Interrupt 58
5.4.8 Display Port Interrupt 58
5.5 Hardware Status Writes 58
5.6 Interrupts 58
5.7 Errors 58
5.7.1 Error Reporting 59
5.7.2 Page Table Errors 59
5.7.3 Clearing Errors 63
5.8 Rendering Context Management 64
5.8.1 Multiple Logical Rendering Contexts 64
5.9 Reset State 67

1. Introduction

The HW supports three engines. The render command streamer interfaces to 3D/IE and display streams. The media command streamer interfaces to the fixed function media. The blitter command streamer interfaces to the blit commands. Software interfaces of all 3 engines are very similar and should only differ on engine-specific functionality.

2. Virtual Memory

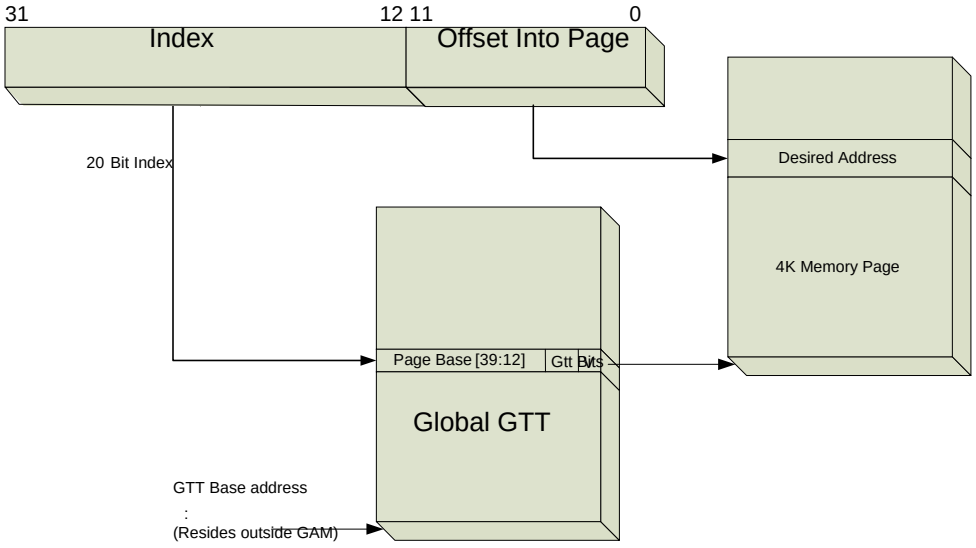
2.1 Virtual Memory Control

SandyBridge [DevSNB] products support different types of graphics translation tables for virtual memory. Global graphics translation table (GGTT) is a single common tranlation table used for all processes. There can be many Per-process graphics translation table (PPGTT). This requires an additional lookup for translation. The actual location is not accessable directly via software since their both located in graphics stolen memory (see graphics memory interface chapter for more detail).

Virtual Memory Structure	[DevSNB]
Global (GGTT)	GSM Only
Per-Process (PPGTT)	2-level, Page Directory in GSM, Page Tables anywhere

2.2 Global Virtual Memory

Global Virtual Memory is the default target memory if a PPGTT is not enabled. If a PPGTT is also present, the method to choose which is targeted by memory and rendering operations varies by product. See the sections on Per-Process Virtual Memory for more information. High priority graphics clients such as Display and Cursor always access global virtual memory.



2.2.1 Graphics Translation Table (GTT) Range (GTTADR)

Address Offset: GTTADR in CPU Physical Space

Access: Aligned DWord Read/Write

The GTTADR memory BAR defined in graphics device config space is an alias for the Global GTT.

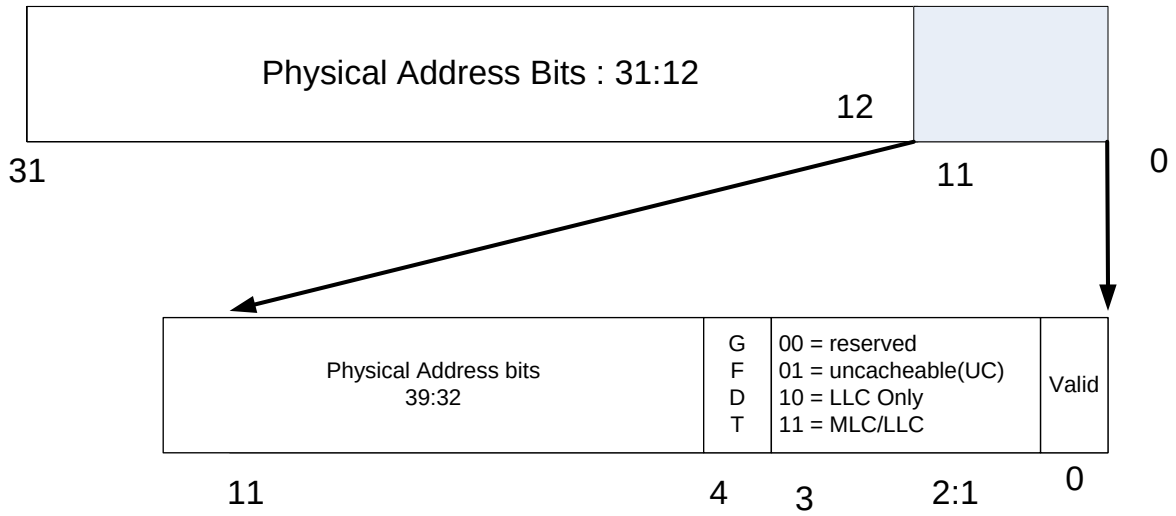
Programming Notes: It is recommended that the driver map all graphics memory pages in the GGTT to some physical page, if only a dummy page.

2.2.2 GTT Page Table Entries (PTEs)

2.2.2.1 GTT Page Table Entries (PTEs) SNB

Page Table Entry: 1 DWord per 4KB Graphics Memory page.

Graphics Page Table Entry Format



Bits	Description
31: 12	**Physical Page Address 31:12:** If the Valid bit is set, This field provides the page number of the physical memory page backing the corresponding Graphics Memory page.
11:4	Physical Start Address Extension: This field specified Bits 39:32 of the page table entry. This field must be zero for 32 bit addresses.

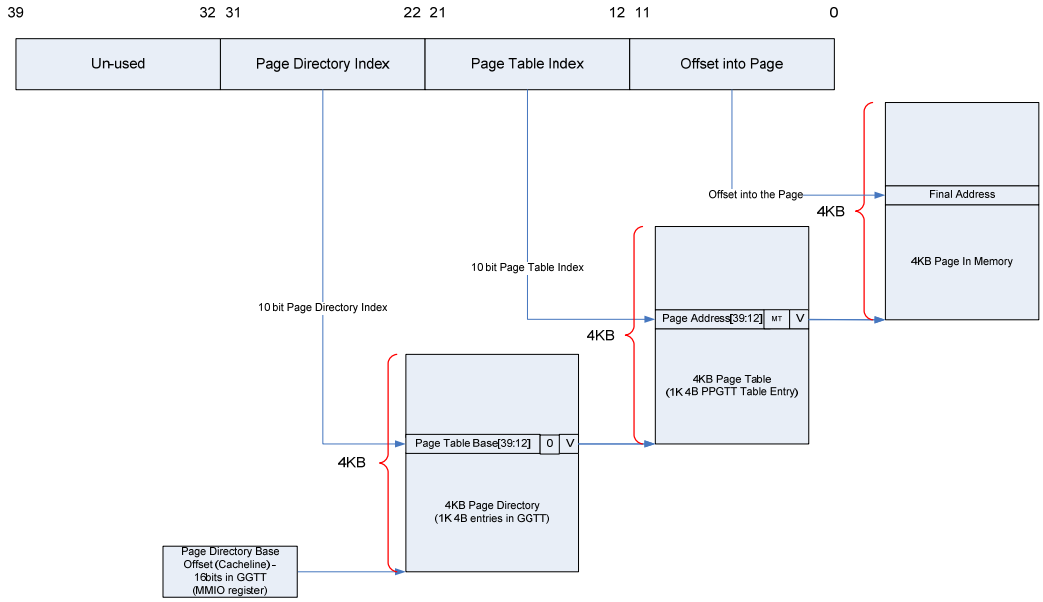
Bits	Description
3	**Graphics Data Type (GFDT)** This field contains the GFDT bit for this surface when writes occur. GFDT can also be set by various state commands and indirect objects. The effective GFDT is the logical OR of the GTT entry with this field. This field is ignored for reads. Format = U1 0: No GFDT (default) 1: GFDT
2	**LLC Cacheability Control (LLCCC)** This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. 0: use cacheability controls from GTT entry 1: Data is cached in LLC
1	**L3 Cacheability Control (L3CC)** This field is used to control the L3 cacheability (allocation) of the stream. 0: not cacheable within L3 1: cacheable in L3 *Note: even if the surface is not cacheable in L3, it is still kept coherent with L3 content.*
0	**Valid PTE:** This field indicates whether the mapping of the corresponding Graphics Memory page is valid. 1: Valid 0: Invalid. An access (other than a CPU Read) through an invalid PTE will result in Page Table Error (Invalid PTE).

2.3 Two-Level Per-Process Virtual Memory

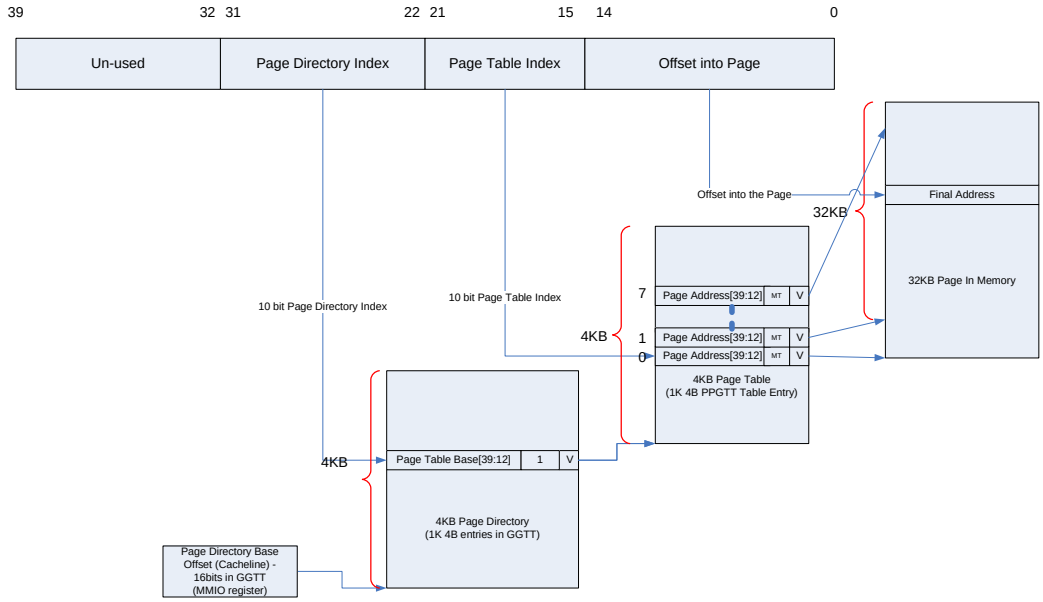
The DevSNB family supports a 2-level mapping scheme for PPGTT, consisting of a first-level page directory containing page table base addresses, and the page tables themselves on the 2nd level, consisting of page addresses. The motivation for the 2-level scheme is simple – it allows for the lookup table (the collection of page tables) to exist in discontiguous memory, making allocation of memory for these structures less problematic for the OS. The directory and each page table fit within a single 4K page of memory that can be located anywhere in physical memory space.

If a PPGTT is enabled, all rendering operations (including blit commands) target Per-process virtual memory. This means all commands *except* the Memory Interface Commands (MI_*). Certain Memory Interface Commands have a specifier to choose global virtual memory (mapped via the GGTT) instead of per-process memory. Global Virtual Memory can be thought of as “privileged” memory in this case. Commands that elect to access privileged memory must have sufficient access rights to do so. Commands executing directly from a ring buffer or from a “secure” batch buffer (see the MI_BATCH_BUFFER_START command in Memory Interface Commands) have these access rights; other commands do not and will not be permitted to access global virtual memory. See the Memory Interface Commands chapters for details on command access of privileged memory.

The PPGTT is disabled by resetting the **Per-Process GTT Enable** bit.



For 32KB Big Pages, the non-big page client the access has no different view, similar to 4KB pages. In case of a big page client, the following access path is used



<u>The starting address of a 32KB page needs to be natively aligned to a 32KB boundary in memory. Hardware uses certain bits (15 and up) to check for the TLB look-ups.</u>

2.3.1 PPGTT Directory Entries (PDEs)

Directory Entry: 1 DWord per 4KB page table (4MB Graphics Memory). Page directories must be located entirely within the GGTT (the table itself.) Directory entries should be updated only for non-active contexts. If a directory entry update is done for a running context, it is unknown when that update will take effect since the device caches directory entries. Directory entries can only be modified using GTTADDR (see *Memory Interface Commands for Rendering Engine*).

31	12	11	4	3	2	1	0
Physical Page Address 31:12		Physical Page Address 39:32		Reserved		Size ("0":4KB, "1":32KB)	Valid

Bits	Description
31:12	**Physical Page Address 31:12**: If the Valid bit is set, This field provides the page number of the physical memory page backing the corresponding Graphics Memory page.
11:8	Reserved: MBZ
7:4	**Physical Page Address Extension:** This field specifies bits 35:32 of the directory entry.
3:1	Reserved: MBZ
1	**Page Size**: Two page sizes are supported thru PDE. **0**: 4KB pages **1**: 32KB pages
0	**Valid PDE:** This field indicates whether this directory entry is valid. 1: Valid 0: Invalid. An access through an invalid PDE will result in a page fault.

2.3.2 PPGTT Table Entries (PTEs)

2.3.2.1 PPGTT Table Entries (PTEs)

Page Table Entry: 1 DWord per 4KB Graphics Memory page. Page Tables must be located in main memory (not snooped). They can be updated directly in memory if proper precautions are taken, or from the command stream by using the MI_UPDATE_GTT command (see *Memory Interface Commands for Rendering Engine*).

31 12	11:4	3	2 1	0
Physical Page Address 31:12	Physical Page Address 39:32	GFDT	Cacheability Control	Valid

Bits	Description
31:12	**Physical Page Address 31:12**: If the Valid bit is set, This field provides the page number of the physical memory page backing the corresponding Graphics Memory page.
11:4	Physical Start Address Extension: This field specified Bits 39:32 of the page table entry. This field must be zero for 32 bit addresses.
3	**Graphics Data Type (GFDT)** This field contains the GFDT bit for this surface when writes occur. GFDT can also be set by various state commands and indirect objects. The effective GFDT is the logical OR of the GTT entry with this field. This field is ignored for reads. Format = U1 0: No GFDT (default) 1: GFDT
2:1	**Cacheability Control** This field controls cacheability in the mid-level cache (MLC) and last-level cache (LLC). Format = U2 enumerated type 00: Reserved 01: data is not cached in LLC 10: data is cached in LLC 11: Reserved
0	**Valid PTE:** This field indicates whether the mapping of the corresponding Graphics Memory page is valid. 1: Valid 0: Invalid. An access (other than a CPU Read) through an invalid PTE will result in Page Table Error (Invalid PTE).

2.3.3 Context TLBs in DevSNB A-step

Due to a bug in DevSNB A-step, the context TLBs are not getting invalidated during a TLB invalidation sequence (i.e. context switching) creating a maintenance/coherency issues for the driver. The workaround of keeping the context pages same is already in place. Other option is to align context GTT entries to CL boundary.

This issue is fixed for B-step onwards.

2.3.4 PPGTT vs GGTT limitation

DevSNB hardware has the limitation to separate the TAGs between PPGTT vs GGTT mapped TLB entries. This requires Virtual Addresses (pages) mapped by PPGTT vs GGTT to be different, else the TLB can mix-match the mappings. Either they need to be different or use same physical mapping for Command Streamer, Vertex Buffers and Instruction Buffers. This limitation is applicable for all the Command Streamer, Vertex Fetch unit, State and Instruction and constant memory spaces.

For DevSNB, s/w will overlap the entire PPGTT and GGTT to work around the issue.

3. GFX MMIO – MCHBAR Aperture

Address Offset:	140000h – 17FFFFh
Default Value:	Same as MCHBAR
Access:	Aligned Word, Dword or Qword Read/Write

This range defined in the graphics MMIO range is an alias with which graphics driver can read and write registers defined in the MCHBAR MMIO space claimed thru Device #0. Attributes for registers defined within the MCHBAR space are preserved when the same registers are accessed via this space. Registers that the graphics driver requires access to are Rank Throttling, GMCH Throttling, Thermal Sensor etc.

The Alias functions works for MMIO access from the CPU only. A command stream load register immediate will drop the data and store register immediate will return all Zeros.

Graphics MMIO registers can be accessed thru MMIO BARs in function #0 and function #1 in Device #2. The aliasing mechanism is turned off if memory access to the corresponding function is turned off via software or in certain power states.

4. Graphics Memory Interface Functions

Note: In the interest of maintainability, functional descriptions of features being added, with the exception of GTT changes, are included in the MI Registers chapter and, to a lesser extent, the *Memory Interface Commands for Rendering* and *Memory Interface Commands for Video Decode* chapters.

4.1 Introduction

The major role of an integrated graphics device's Memory Interface (MI) function is to provide various client functions access to "graphics" memory used to store commands, surfaces, and other information used by the graphics device. This chapter describes the basic mechanisms and paths by which graphics memory is accessed.

Information not presented in this chapter includes:

Microarchitectural and implementation-dependent features (e.g., internal buffering, caching and arbitration policies).

MI functions and paths specific to the operation of external (discrete) devices attached via external connections.

MI functions essentially unrelated to the operation of the internal graphics devices, .e.g., traditional "chipset functions" (refer to the device's C-Spec for this information).

4.2 Graphics Memory Clients

The MI function provides memory access functionality to a number of external and internal graphics memory *clients*, as described in Table 4-1.

Table 4-1. Graphics Memory Clients

MI Client	Access Modes
Host Processor	Read/Write of Graphics Operands located in Main Memory. Graphics Memory is accessed using Device 2 Graphics Memory Range Addresses
External PEG Graphics Device	**Write-Only** of Graphics Operands located in Main Memory via the Graphics Aperture. (This client is not described in this chapter).
Peer PCI Device	**Write-Only** of Graphics Operands located in Main Memory. Graphics Memory is accessed using Device 2 Graphics Memory Range Addresses (i.e., mapped by GTT*). Note that DMI access to Graphics registers is not supported.*
Snooped Read/Write (internal)	Internally-generated snooped reads/writes.
Command Stream (internal)	DMA Read of graphics commands and related graphics data.
Vertex Stream (internal)	DMA Read of indexed vertex data from Vertex Buffers by the 3D Vertex Fetch (VF) Fixed Function.

MI Client	Access Modes
Instruction/State Cache (internal)	Read of pipelined 3D rendering state used by the 3D/Media Functions and instructions executed by the EUs.
Render Cache (internal)	Read/Write of graphics data operated upon by the graphics rendering engines (Blt, 3D, MPEG, etc.) Read of render surface state.
Sampler Cache (internal)	Read of texture (and other sampled surface) data stored in graphics memory.
Display/Overlay Engines (internal)	Read of display, overlay, cursor and VGA data.

4.3 Graphics Memory Addressing Overview

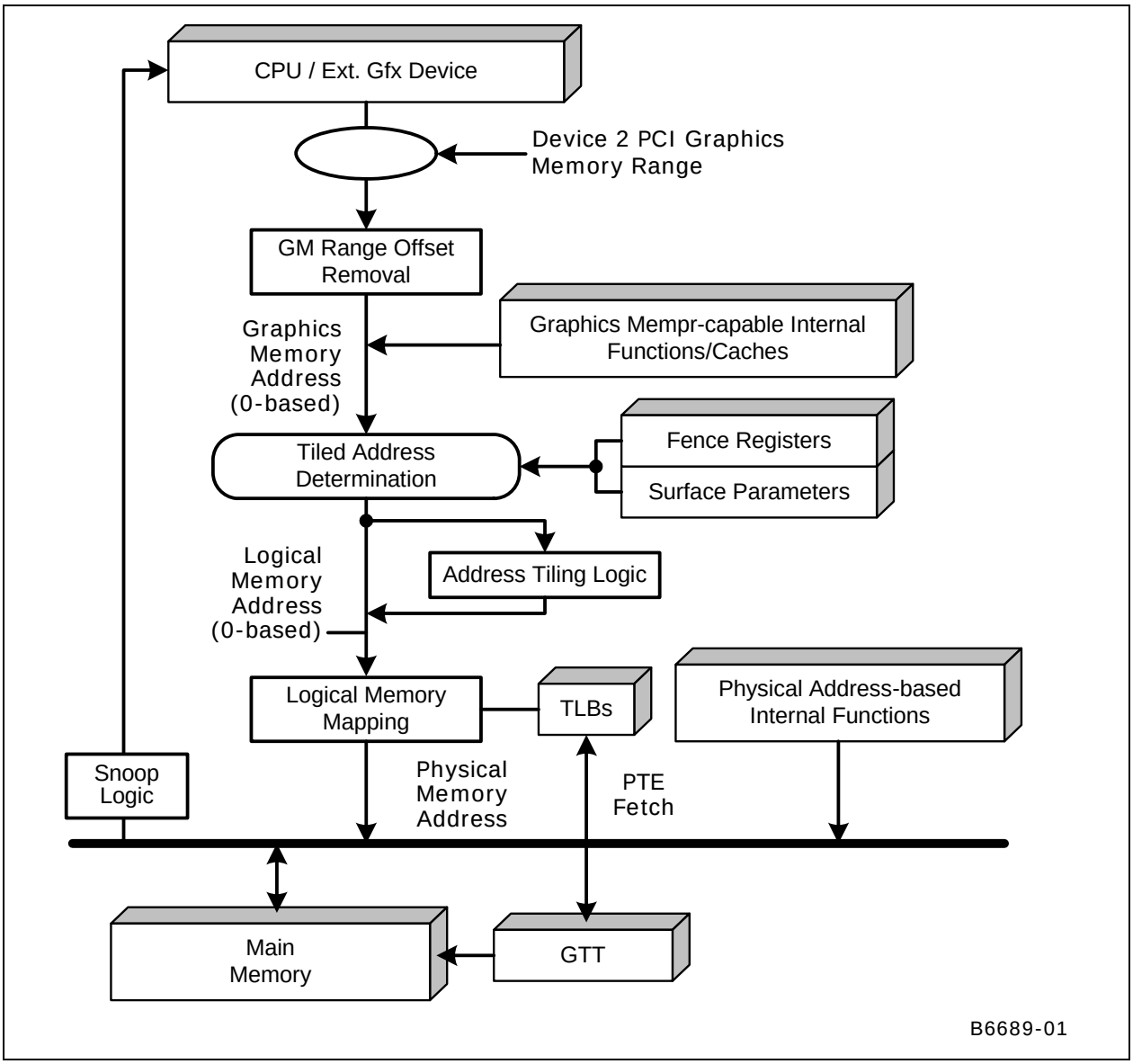
The Memory Interface function provides access to graphics memory (GM) clients. It accepts memory addresses of various types, performs a number of optional operations along *address paths*, and eventually performs reads and writes of graphics memory data using the resultant addresses. The remainder of this subsection will provide an overview of the graphics memory clients and address operations.

4.3.1 Graphics Address Path

Figure 4-1 shows the internal graphics memory address path, connection points, and optional operations performed on addresses. Externally-supplied addresses are normalized to zero-based *Graphics Memory (GM) addresses* (GM_Address). If the GM address is determined to be a tiled address (based on inclusion in a fenced region or via explicit surface parameters), *address tiling* is performed. At this point the address is considered a *Logical Memory address*, and is translated into a *Physical Memory address* via the GTT and associated TLBs. The physical memory location is then accessed.

CPU accesses to graphics memory are not snooped on the front side bus post GTT translation. Hence pages that are mapped cacheable in the GTT will not be coherent with the CPU cache if accessed through graphics memory aperture. Also, such accesses may have side effects in the hardware.

Figure 4-1. Graphics Memory Paths



The remainder of this chapter describes the basic features of the graphics memory address pipeline, namely Address Tiling, Logical Address Mapping, and Physical Memory types and allocation considerations.

4.4 Graphics Memory Address Spaces

Table 4-2 lists the five supported Graphics Memory Address Spaces. Note that the Graphics Memory Range Removal function is automatically performed to transform system addresses to internal, zero-based Graphics Addresses.

Table 4-2. Graphics Memory Address Types

Address Type	Description	Range
Dev2_GM_Address	Address range allocated via the Device 2 (integrated graphics device) GMADR register. The processor and other peer (DMI) devices utilize this address space to read/write graphics data that resides in Main Memory. This address is internally converted to a GM_Address.	Some 64MB, 128MB, 256MB or 512MB address range normally above TOM
GM_Address	Zero-based logical Graphics Address, utilized by internal device functions to access GTT-mapped graphics operands. GM_Addresses are typically passed in commands and contained in state to specify operand location.	[0, 64MB-1], [0, 128MB-1], [0, 256MB-1] or [0, 512MB-1] [0, 1GB-1] on [DevCTG] Only
PGM_Address	Zero-based logical Per-Process Graphics Address, utilized by internal device functions to access render GTT (PPGTT) mapped graphics operands. Memory in this space is not accessible by the processor and other peer (DMI) devices unless aliased to a GM_Address.	[0, 64MB-1], [0,128MB-1], [0,256MB-1], [0,512MB-1] or [0, 1GB – 1]

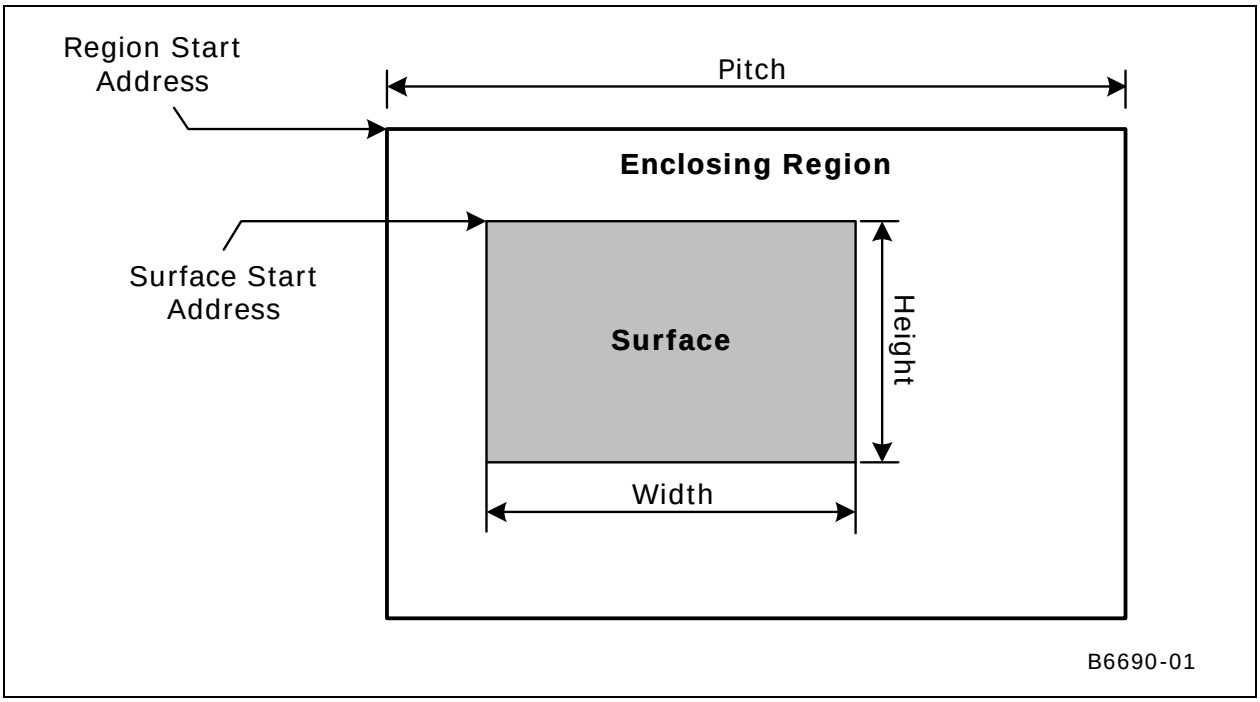
4.5 Address Tiling Function

When dealing with memory operands (e.g., graphics surfaces) that are inherently rectangular in nature, certain functions within the graphics device support the storage/access of the operands using alternative (tiled) memory formats in order to increase performance. This section describes these memory storage formats, why/when they should be used, and the behavioral mechanisms within the device to support them.

4.5.1 Linear vs. Tiled Storage

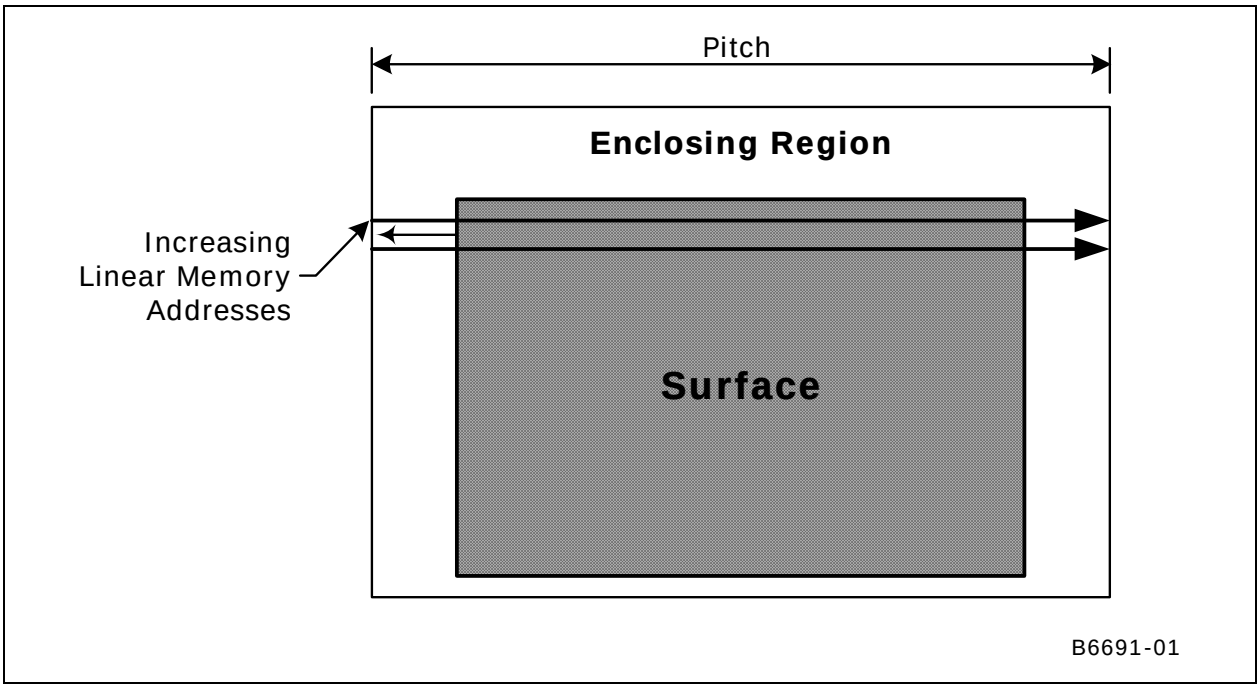
Regardless of the memory storage format, “rectangular” memory operands have a specific *width* and *height*, and are considered as residing within an enclosing rectangular region whose width is considered the *pitch* of the region and surfaces contained within. Surfaces stored within an enclosing region must have widths less than or equal to the region pitch (indeed the enclosing region may coincide exactly with the surface). Figure 4-2 shows these parameters.

Figure 4-2. Rectangular Memory Operand Parameters



The simplest storage format is the *linear* format (see Figure 4-3), where each row of the operand is stored in sequentially increasing memory locations. If the surface width is less than the enclosing region's pitch, there will be additional memory storage between rows to accommodate the region's pitch. The pitch of the enclosing region determines the distance (in the memory address space) between vertically-adjacent operand elements (e.g., pixels, texels).

Figure 4-3. Linear Surface Layout

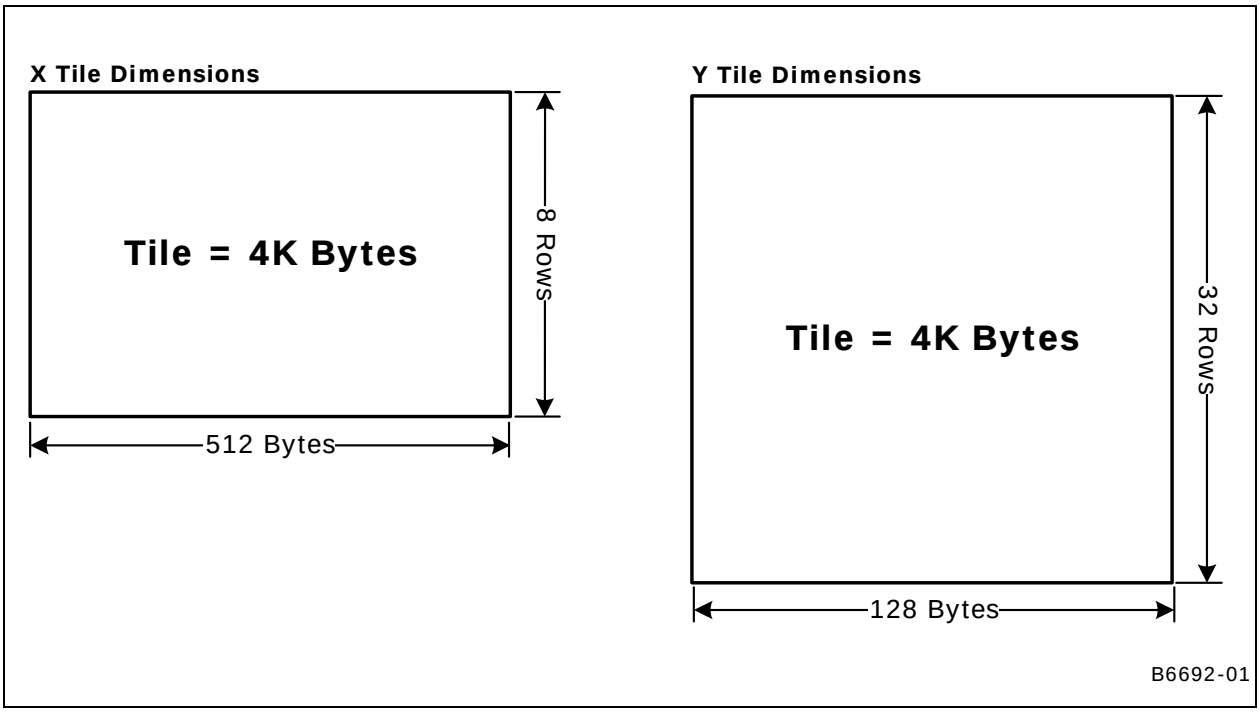


The linear format is best suited for 1-dimensional row-sequential access patterns (e.g., a display surface where each scanline is read sequentially). Here the fact that one object element may reside in a different memory page than its vertically-adjacent neighbors is not significant; all that matters is that horizontally-adjacent elements are stored contiguously. However, when a device function needs to access a 2D subregion within an operand (e.g., a read or write of a 4x4 pixel span by the 3D renderer, a read of a 2x2 texel block for bilinear filtering), having vertically-adjacent elements fall within different memory pages is to be avoided, as the page crossings required to complete the access typically incur increased memory latencies (and therefore lower performance).

One solution to this problem is to divide the enclosing region into an array of smaller rectangular regions, called memory *tiles*. Surface elements falling within a given tile will all be stored in the same physical memory page, thus eliminating page-crossing penalties for 2D subregion accesses within a tile and thereby increasing performance.

Tiles have a fixed 4KB size and are aligned to physical DRAM page boundaries. They are either 8 rows high by 512 bytes wide or 32 rows high by 128 bytes wide (see Figure 4-4). Note that the dimensions of tiles are irrespective of the data contained within – e.g., a tile can hold twice as many 16-bit pixels (256 pixels/row x 8 rows = 2K pixels) than 32-bit pixels (128 pixels/row x 8 rows = 1K pixels).

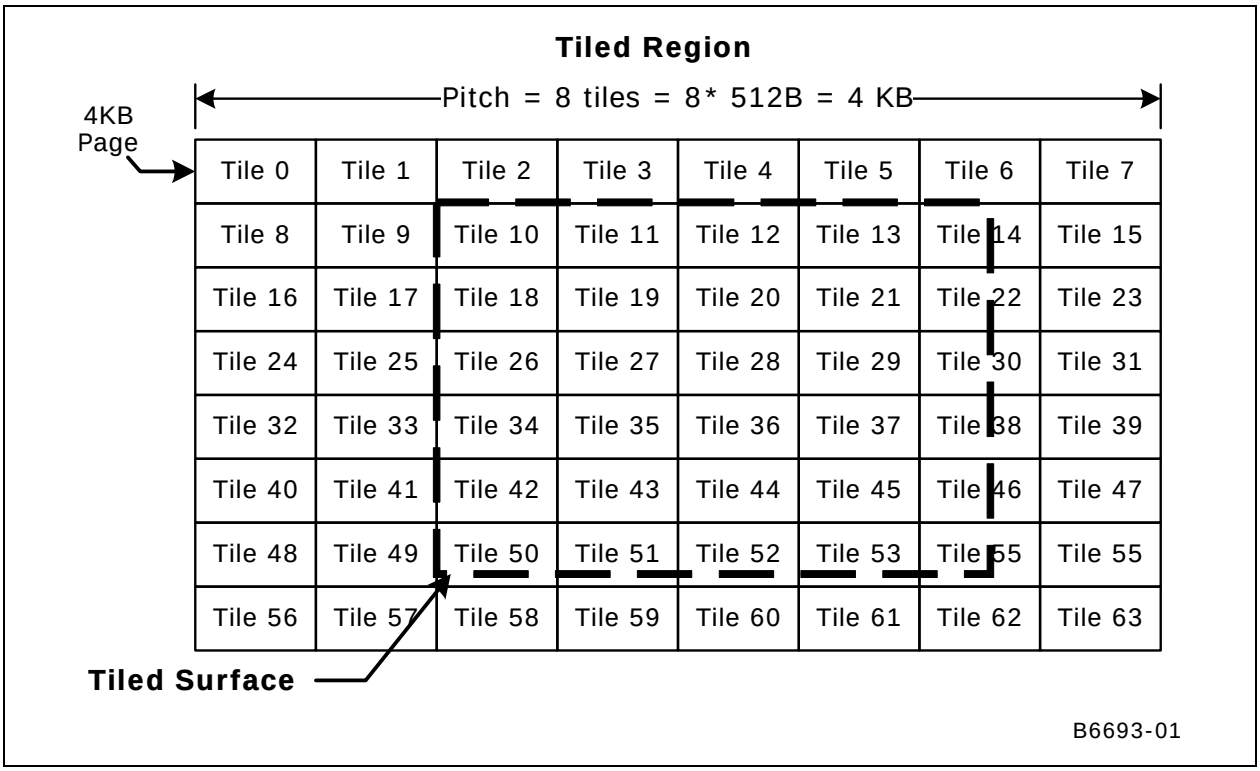
Figure 4-4. Memory Tile Dimensions



The pitch of a tiled enclosing region must be an integral number of tile widths. The 4KB tiles within a tiled region are stored sequentially in memory in row-major order.

Figure 4-5 shows an example of a tiled surface located within a tiled region with a pitch of 8 tile widths (512 bytes * 8 = 4KB). Note that it is the enclosing region that is divided into tiles – the surface is not necessarily aligned or dimensioned to tile boundaries.

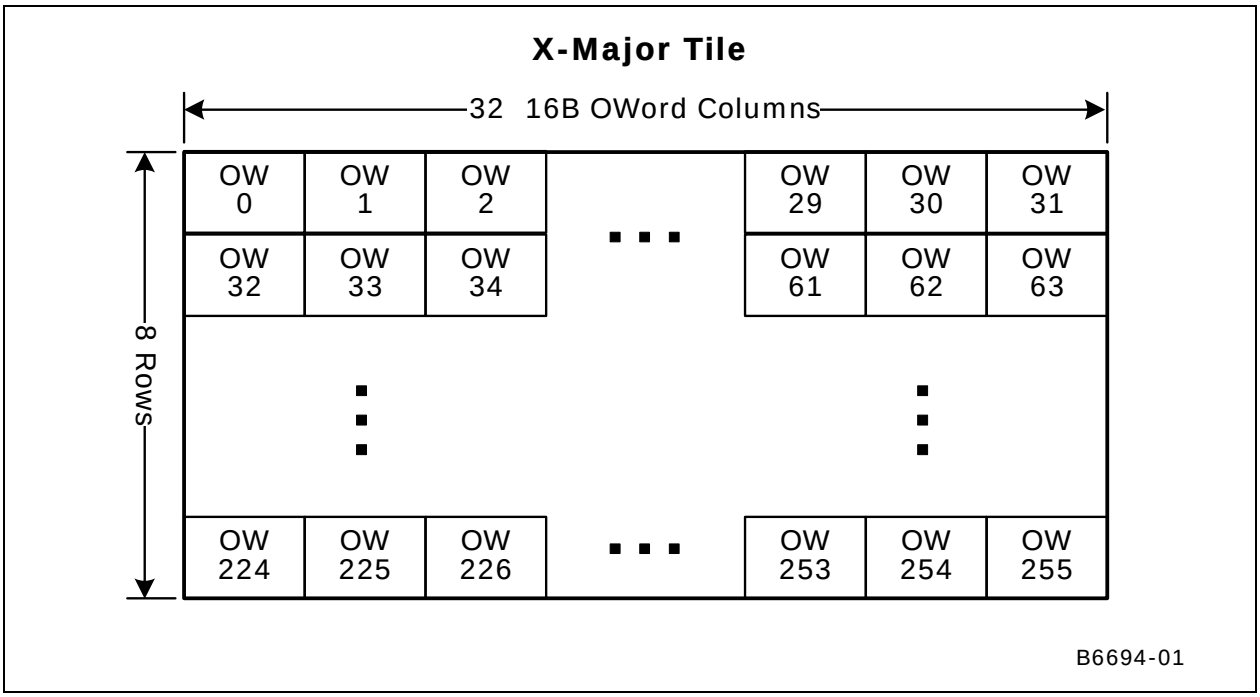
Figure 4-5. Tiled Surface Layout



4.5.2 Tile Formats

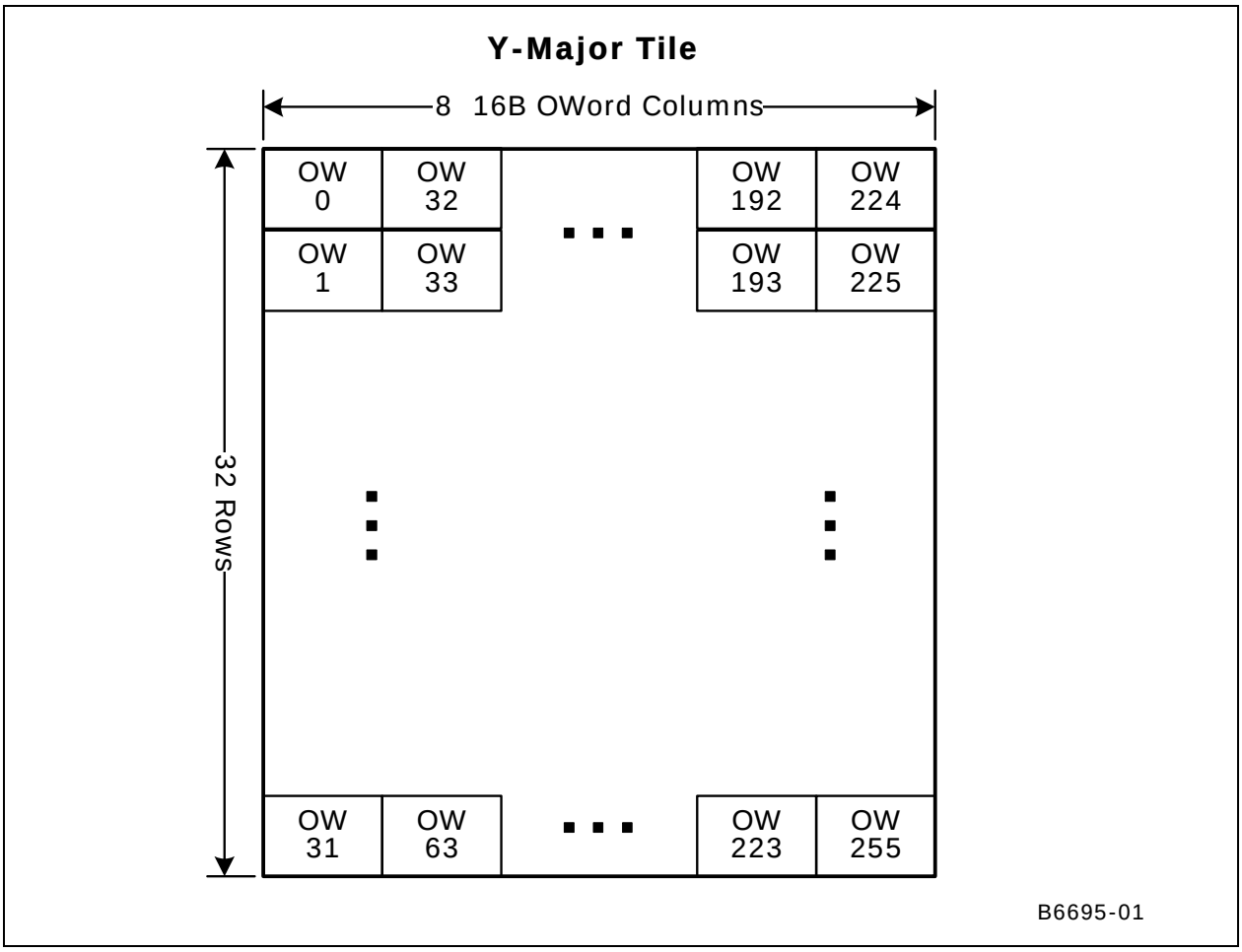
The device supports both *X-Major* (row-major) and *Y-Major* (column major) storage of tile data units, as shown in the following figures. A 4KB tile is subdivided into an 8-high by 32-wide array of 16-byte OWords for X-Major Tiles (X Tiles for short), and 32-high by 8-wide array of OWords for Y-Major Tiles (Y Tiles). The selection of tile direction only impacts the internal organization of tile data, and does not affect how surfaces map onto tiles. Note that the diagrams are not to scale – the first format defines the contents of an 8-high by 512-byte wide tile, and the 2nd a 32-high by 128-byte wide tile. The storage of tile data units in X-Major or Y-Major fashion is sometimes refer to as the *walk* of the tiling.

Table 4-3. X-Major Tile Layout



Note that an X-major tiled region with a tile pitch of 1 tile is actually stored in a linear fashion.

Figure 4-6. Y-Major Tile Layout



4.5.2.1 W-Major Tile Format

The device supports additional format *W-Major* storage of tile data units, as shown in the following figures. A 4KB tile is subdivided into 8-high by 8-wide array of Blocks for W-Major Tiles (W Tiles). Each Block is 8 rows by 8 bytes. The selection of tile direction only impacts the internal organization of tile data, and does not affect how surfaces map onto tiles. W-Major Tile Format is used for separate stencil.

Figure 4-7. W-Major Tile Layout

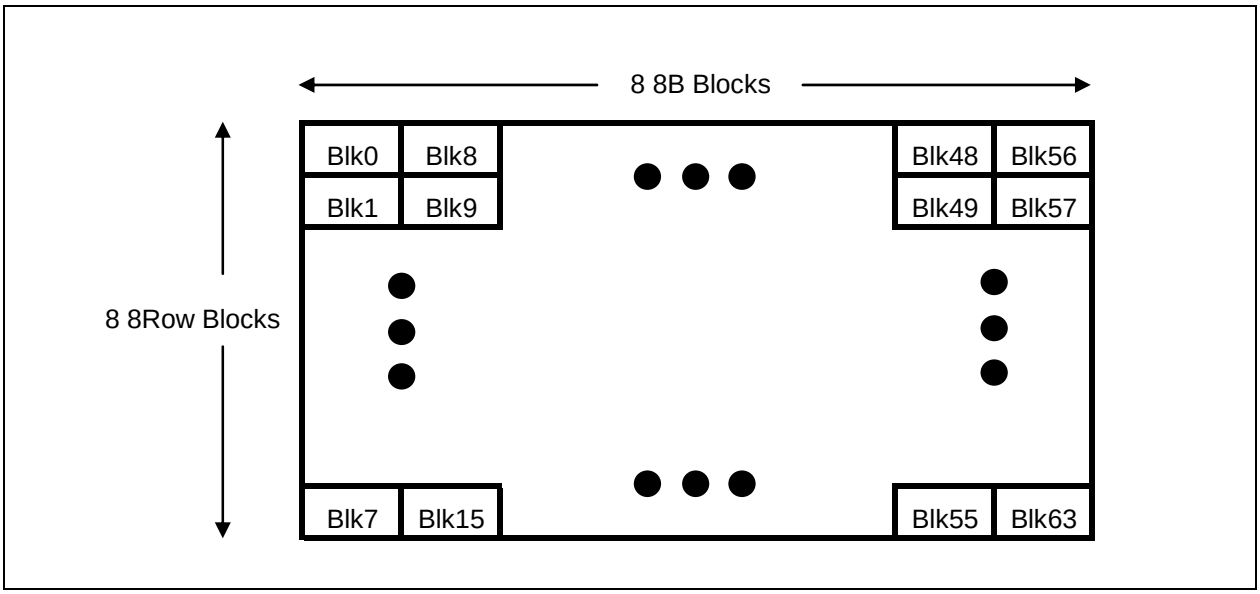
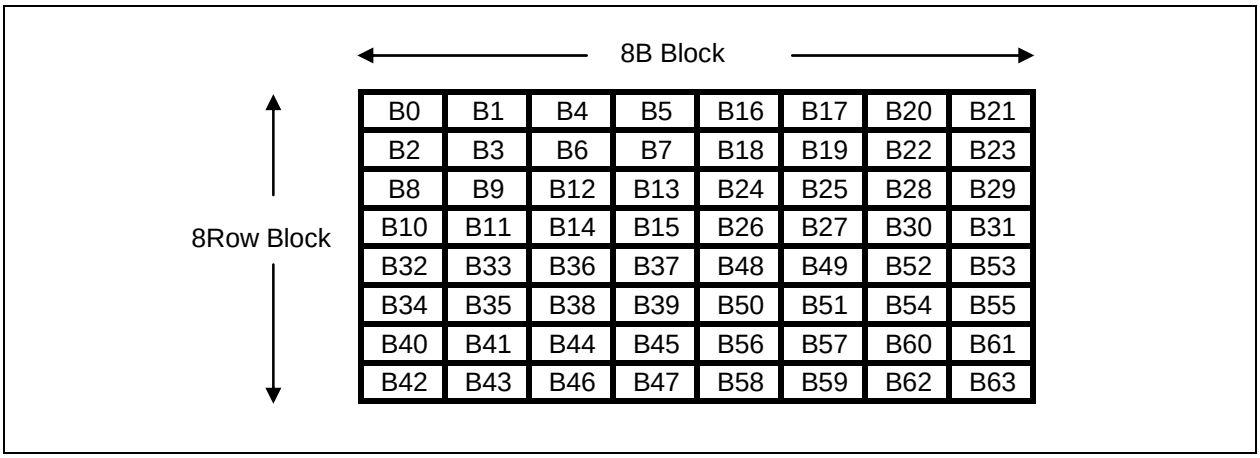


Figure 4-8. W-Major Block Layout



4.5.3 Tiling Algorithm

The following pseudocode describes the algorithm for translating a tiled memory surface in graphics memory to an address in logical space.

```
Inputs: LinearAddress(offset into aperture in terms of bytes),
      Pitch(in terms of tiles),
      WalkY (1 for Y and 0 for X)
      WalkW (1 for W and 0 for the rest)

Static Parameters:
      TileH (Height of tile, 8 for X, 32 for Y and 64 for W),
      TileW (Width of Tile in bytes, 512 for X, 128 for Y and 64 for W)


TileSize = TileH * TileW;
RowSize = Pitch * TileSize;


If (Fenced) {

      LinearAddress = LinearAddress – FenceBaseAddress

      LinearAddrInTileW = LinearAddress div TileW;

      Xoffset_inTile = LinearAddress mod TileW;

      Y = LinearAddrInTileW div Pitch;

      X = LinearAddrInTileW mod Pitch + Xoffset_inTile;

}


// Internal graphics clients that access tiled memory already have the
X, Y
// coordinates and can start here

YOff_Within_Tile = Y mod TileH;

XOff_Within_Tile = X mod TileW;


TileNumber_InY = Y div TileH;

TileNumber_InX = X div TileW;
```

```
TiledOffsetY = RowSize * TileNumber_InY + TileSize * TileNumber_InX +
            TileH * 16 * (XOff_Within_Tile div 16) + YOff_Within_Tile *
            16 + (XOff_Within_Tile mod 16);


TiledOffsetW = RowSize * TileNumber_InY + TileSize * TileNumber_InX +
      TileH * 8 * (XOff_Within_Tile div 8) +
      64* (YOff_Within_Tile div 8)+
      32*((YOff_Within_Tile div 4) mod 2) +
      16* ((XOff_Within_Tile div 4) mod 2) +
      8 * ((YOff_Within_Tile div 2) mod 2) +
      4* ((XOff_Within_Tile div 2) mod 2) +
      2 * (YOff_Within_Tile mod 2) +
      (XOff_Within_Tile mod 2);


TiledOffsetX = RowSize * TileNumber_InY + TileSize * TileNumber_InX +
            TileW * YOff_Within_Tile + XOff_Within_Tile;


TiledOffset = WalkW? TiledOffsetW : (WalkY? TiledOffsetY :
TiledOffsetX);




TiledAddress = Tiled? (BaseAddress + TiledOffset): (BaseAddress +
            Y*LinearPitch + X);

}
```

The Y-Major tile formats have the characteristic that a surface element in an even row is located in the same aligned 64-byte cacheline as the surface element immediately below it (in the odd row). This spatial locality can be exploited to increase performance when reading 2x2 texel squares for bilinear texture filtering, or reading and writing aligned 4x4 pixel spans from the 3D Render pipeline.

On the other hand, the X-Major tile format has the characteristic that horizontally-adjacent elements are stored in sequential memory addresses. This spatial locality is advantageous when the surface is scanned in row-major order for operations like display refresh. For this reason, the Display and Overlay memory streams only support linear or X-Major tiled surfaces (Y-Major tiling is not supported by these functions). This has the side effect that 2D- or 3D-rendered surfaces must be stored in linear or X-Major tiled formats if they are to be displayed. Non-displayed surfaces, e.g., “rendered textures”, can also be stored in Y-Major order.

4.5.4 Tiling Support

The rearrangement of the surface elements in memory must be accounted for in device functions operating upon tiled surfaces. (Note that not all device functions that access memory support tiled formats). This requires either the modification of an element's linear memory address or an alternate formula to convert an element's X,Y coordinates into a tiled memory address.

However, before tiled-address generation can take place, some mechanism must be used to determine whether the surface elements accessed fall in a linear or tiled region of memory, and if tiled, what the tile region pitch is, and whether the tiled region uses X-Major or Y-Major format. There are two mechanisms by which this detection takes place: (a) an implicit method by detecting that the pre-tiled (linear) address falls within a "fenced" tiled region, or (b) by an explicit specification of tiling parameters for surface operands (i.e., parameters included in surface-defining instructions).

The following table identifies the tiling-detection mechanisms that are supported by the various memory streams.

Access Path	Tiling-Detection Mechanisms Supported
Processor access through the Graphics Memory Aperture	Fenced Regions
3D Render (Color/Depth Buffer access)	Explicit Surface Parameters
Sampled Surfaces	Explicit Surface Parameters
Blt operands	Explicit Surface Parameters
Display and Overlay Surfaces	Explicit Surface Parameters

4.5.4.1 Tiled (Fenced) Regions

The only mechanism to support the access of surfaces in tiled format by the host or external graphics client is to place them within "fenced" tiled regions within Graphics Memory. A fenced region is a block of Graphics Memory specified using one of the sixteen FENCE device registers. (See *Memory Interface Registers* for details). Surfaces contained within a fenced region are considered tiled from an external access point of view. Note that fences cannot be used to untile surfaces in the PGM_Address space since external devices cannot access PGM_Address space. Even if these surfaces (or any surfaces accessed by an internal graphics client) fall within a region covered by an enabled fence register, that enable will be effectively masked during the internal graphics client access. Only the explicit surface parameters described in the next section can be used to tile surfaces being accessed by the internal graphics clients.

Each FENCE register (if its Fence Valid bit is set) defines a Graphics Memory region ranging from 4KB to the aperture size. The region is considered rectangular, with a pitch in tile widths from 1 tile width (128B or 512B) to 256 tile X widths (256 * 512B = 128KB) and 1024 tile Y widths (1024 * 128B = 128KB). Note that fenced regions must not overlap, or operation is UNDEFINED.

Also included in the FENCE register is a Tile Walk field that specifies which tile format applies to the fenced region.

4.5.4.1.1 FENCE — Graphics Memory Fence Table Registers

FENCE — Graphics Memory Fence Table Registers

Register Type:	MMIO
Address Offset:	100000h
Project:	All
Default Value:	00000000h;
Access:	R/W
Size (in bits):	16x64
Trusted Type:	1

Address Offset: 00100000h – 001000007h: FENCE_0

:

:

00100078h – 0010007Fh: FENCE_15

The graphics device performs address translation from linear space to tiled space for a CPU access to graphics memory (See *Memory Interface Functions* chapter for information on these memory layouts) using the fence registers. Note that the fence registers are used **only for CPU accesses to gfx memory**. Graphics rendering/display pipelines use Per Surface Tiling (PST) parameters (found in SURFACE_STATE – see the *Sampling Engine* chapter) to access tiled gfx memory.

The intent of tiling is to locate graphics data that are close (in X and Y surface axes) in one physical memory page while still locating some amount of line oriented data sequentially in memory for display efficiency. All 3D rendering is done such that the QWords of any one span are all located in the same memory page, improving rendering performance. Applications view surfaces as linear, hence when the cpu access a surface that is tiled, the gfx hardware must perform linear to tiled address conversion and access the correct physical memory location(s) to get the data.

Tiled memory is supported for rendering and display surfaces located in graphics memory. A tiled memory surface is a surface that has a width and height that are subsets of the tiled region's pitch and height. The device maintains the constants required by the memory interface to perform the address translations. Each tiled region can have a different pitch and size. The CPU-memory interface needs the surface pitch and tile height to perform the address translation. It uses the GMAddr (PCI-BAR) offset address to compare with the fence start and end address, to determine if the rendering surface is tiled. The tiled address is generated based on the tile orientation determined from the matching fence register. Fence ranges are at least 4 KB aligned. Note that the fence registers are used <u>only for CPU accesses</u> to graphics memory.

A Tile represents 4 KB of memory. Tile height is 8 rows for X major tiles and 32 rows for Y major tiles. Tile Pitch is 512Bs for X major tiles and 128Bs for Y major tiles. The surface pitch is programmed in 128B units such that the pitch is an integer multiple of "tile pitch".

Engine restrictions on tile surface usage are detailed in Surface Placement Restrictions (Memory Interface Functions). Note that X major tiles can be used for Sampler, Color, Depth, motion compensation references and motion compensation destination, Display, Overlay, GDI Blt source and destination surfaces. Y major tiles can be used for Sampler, depth, color and motion compensation assuming they do not need to be displayed. GDI Blit operations, overlay and display cannot used Tiled Y orientations.

A "PST" graphics surface that will also be accessed via fence needs its base address to be tile row aligned.

FENCE — Graphics Memory Fence Table Registers

Hardware handles the flushing of any pending cycles when software changes the fence upper/lower bounds.

Fence Table Registers occupy the address range specified above. Each Fence Table Register has the following format.

FENCE registers are *not* reset by a graphics reset. They will maintain their values unless a full chipset reset is performed.

DWord	Bit	Description
0..15	63:44	**Fence Upper Bound** Project: All Address: GraphicsAddress[31:12] Bits 31:12 of the ending Graphics Address of the fence region. Fence regions must be aligned to a 4KB page. This address represents the last 4KB page of the fence region (Upper Bound is included in the fence region). Graphics Address is the offset within GMADR space.
	41:32	**Fence Pitch** Project: All Default Value: 0h DefaultVaueDesc Format: U10-1 Width in 128 bytes This field specifies the width (pitch) of the fence region in multiple of "tile width". For Tile X this field must be programmed to a multiple of 512B ("003" is the minimum value) and for Tile Y this field must be programmed to a multiple of 128B ("000" is the minimum value). 000h = 128B 001h = 256B ... 3FFh = 128KB
	31:12	**Fence Lower Bound** Project: All Address: GraphicsAddress[31:12] Bits 31:12 of the starting Graphics Address of the fence region. Fence regions must be aligned to 4KB. This address represents the first 4KB page of the fence region (Lowe Bound is included in the fence region). Graphics Address is the offset within GMADR space.
	11:2	**Reserved** Project: All Format: MBZ

FENCE — Graphics Memory Fence Table Registers

	1	**Tile Walk** Project: All Format: MI_TileWalk This field specifies the spatial ordering of QWords within tiles.
	0	**Fence Valid** Project: All Format: MI_ FenceValid This field specifies whether or not this fence register defines a fence region.

Value	Name	Description	Project
0h	MI_TILE_XMAJOR	Consecutive SWords (32 Bytes) sequenced in the X direction	All
1h	MI_TILE_YMAJOR	Consecutive OWords (16 Bytes) sequenced in the Y direction	All

Value	Name	Description	Project
0h	MI_FENCE_INVALID		All
1h	MI_FENCE_VALID		All

4.5.4.2 Tiled Surface Parameters

Internal device functions require explicit specification of surface tiling parameters via information passed in commands and state. This capability is provided to limit the reliance on the fixed number of fence regions.

The following table lists the surface tiling parameters that can be specified for 3D Render surfaces (Color Buffer, Depth Buffer, Textures, etc.) via SURFACE_STATE.

Surface Parameter	Description
Tiled Surface	If ENABLED, the surface is stored in a tiled format. If DISABLED, the surface is stored in a linear format.
Tile Walk	If Tiled Surface is ENABLED, this parameter specifies whether the tiled surface is stored in Y-Major or X-Major tile format.
Base Address	Additional restrictions apply to the base address of a Tiled Surface vs. that of a linear surface.
Pitch	Pitch of the surface. Note that, if the surface is tiled, this pitch must be a multiple of the tile width.

4.5.4.3 Tiled Surface Restrictions

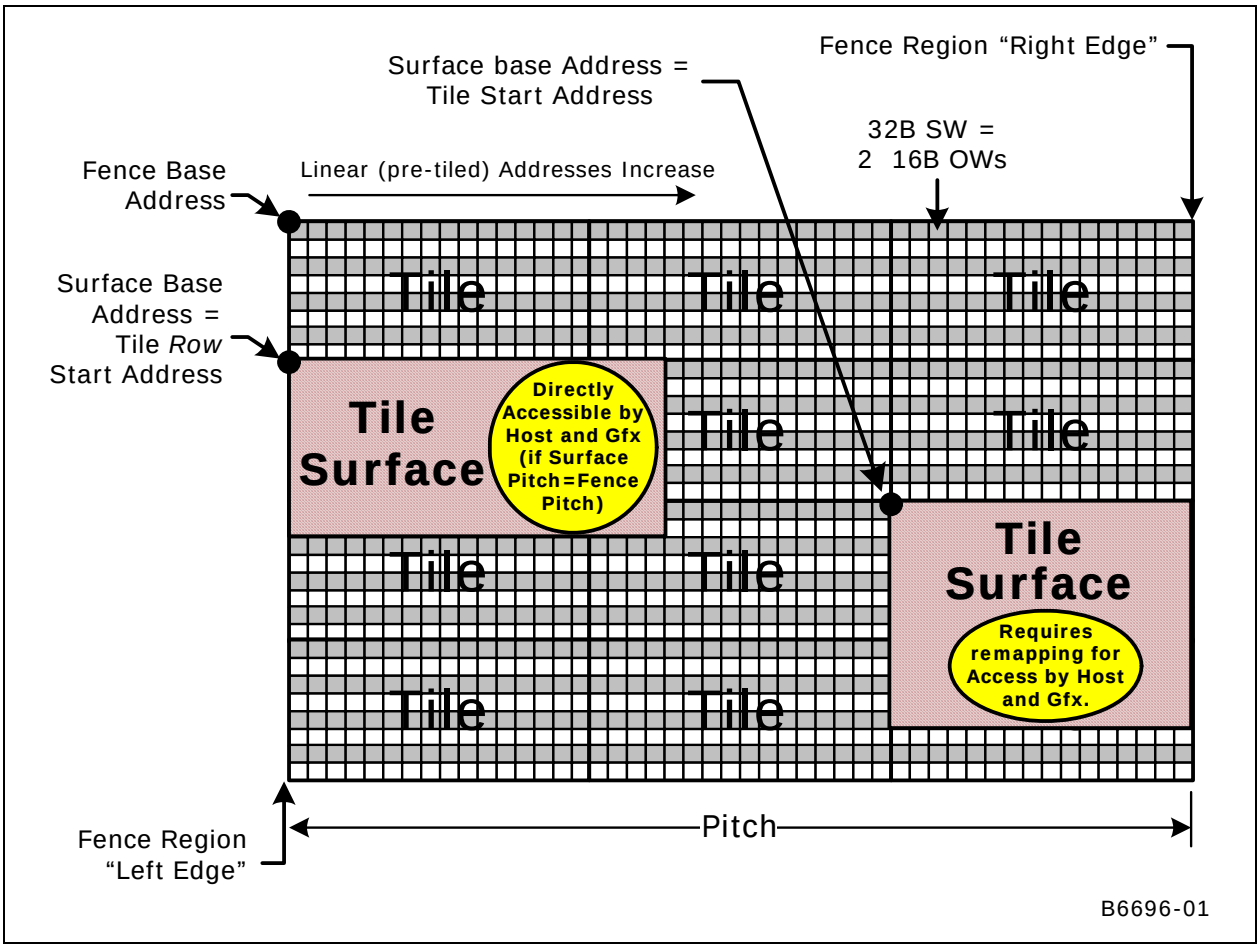
Additional restrictions apply to the Base Address and Pitch of a surface that is tiled. In addition, restrictions for tiling via SURFACE_STATE are subtly different from those for tiling via fence regions. The most restricted surfaces are those that will be accessed both by the host (via fence) and by internal device functions. An example of such a surface is a tiled texture that is initialized by the CPU and then sampled by the device.

The tiling algorithm for internal device functions is different from that of fence regions. Internal device functions always specify tiling in terms of a surface. The surface must have a base address, and this base address is not subject to the tiling algorithm. Only *offsets* from the base address (as calculated by X, Y addressing within the surface) are transformed through tiling. The base address of the surface must therefore be 4KB-aligned. This forces the 4KB tiles of the tiling algorithm to exactly align with 4KB device pages once the tiling algorithm has been applied to the offset. The width of a surface must be less than or equal to the surface pitch. There are additional considerations for surfaces that are also accessed by the host (via a fence region).

Fence regions have no base address per se. Host linear addresses that fall in a fence region are translated in their entirety by the tiling algorithm. It is as if the surface being tiled by the fence region has a base address in graphics memory equal to the fence base address, and all accesses of the surfaces are (possibly quite large) offsets from the fence base address. Fence regions have a virtual “left edge” aligned with the fence base address, and a “right edge” that results from adding the fence pitch to the “left edge”. Surfaces in the fence region must not straddle these boundaries.

Base addresses of surfaces that are to be accessed both by an internal graphics client and by the host have the tightest restrictions. In order for the surface to be accessed without GTT re-mapping, the surface base address (as set in SURFACE_STATE) must be a “Tile Row Start Address” (TRSA). The first address in each tile row of the fence region is a Tile Row Start Address. The first TRSA is the fence base address. Each TRSA can be generated by adding an integral multiple of the row size to the fence base address. The row size is simply the fence pitch in tiles multiplied by 4KB (the size of a tile.)

Figure 4-9. Tiled Surface Placement



The pitch in SURFACE_STATE must be set equal to the pitch of the fence that will be used by the host to access the surface if the same GTT mapping will be used for each access. If the pitches differ, a different GTT mapping must be used to eliminate the "extra" tiles (4KB memory pages) that exist in the excess rows at the right side of the larger pitch. Obviously no part of the surface that will be accessed can lie in pages that exist only in one mapping but not the other. The new GTT mapping can be done manually by SW between the time the host writes the surface and the device reads it, or it can be accomplished by arranging for the client to use a different GTT than the host (the PPGTT -- see Logical Memory Mapping below).

The width of the surface (as set in SURFACE_STATE) must be less than or equal to both the surface pitch and the fence pitch in any scenario where a surface will be accessed by both the host and an internal graphics client. Changing the GTT mapping will not help if this restriction is violated.

Surface Access	Base Address	Pitch	Width	Tile "Walk"
Host only	No restriction	Integral multiple of tile size <= 128KB	Must be <= Fence Pitch	No restriction
Client only	4KB-aligned	Integral multiple of tile size <= 256KB	Must be <= Surface Pitch	Restrictions imposed by the client (see Per-Stream Tile Format Support)
Host and Client, No GTT Remapping	Must be TRSA	Fence Pitch = Surface Pitch = integral multiple of tile size <= 256KB	Width <= Pitch	Surface Walk must meet client restriction, Fence Walk = Surface Walk
Host and Client, GTT Remapping	4KB-aligned for client (will be tile-aligned for host)	Both must be Integral multiple of tile size <=128KB, but not necessarily the same	Width <= Min(Surface Pitch, Fence Pitch)	Surface Walk must meet client restriction, Fence Walk = Surface Walk

4.5.5 Per-Stream Tile Format Support

<table>
<tr><th>MI Client</th><th>Tile Formats Supported</th></tr>
<tr><td>CPU Read/Write</td><td>All</td></tr>
<tr><td>Display/Overlay</td><td>Y-Major not supported.
X-Major required for Async Flips</td></tr>
<tr><td>Blt</td><td>Linear and X-Major only
No Y-Major support</td></tr>
<tr><td>3D Sampler</td><td>All Combinations of TileY, TileX and Linear are supported. TileY is the fastest, Linear is the slowest.</td></tr>
<tr><td>3D Color,Depth</td><td>
<table>
<tr><th>Rendering Mode
Color-vs-Depth bpp</th><th>Buffer Tiling Supported</th></tr>
<tr><td>Classical
Same Bpp</td><td>Both Linear
Both TileX
Both TileY
Linear & TileX
Linear & TileY
TileX & TileY</td></tr>
<tr><td>Classical
Mixed Bpp</td><td>Both Linear
Both TileX
Both TileY
Linear & TileX
Linear & TileY
TileX & TileY</td></tr>
</table>
NOTE: 128BPE Format Color buffer (render target) MUST be either TileX or Linear.
</td></tr>
</table>

4.6 Logical Memory Mapping

In order to provide a contiguous address space for graphics operands (surfaces, etc.) yet allow this address space to be mapped onto possibly discontiguous physical memory pages, the internal graphics device supports a Logical Memory Space (see Figure 4-12). A global *Graphics Translation Table* (GTT) is provided to map zero-based (and post-tiled) Logical Memory Addresses into a set of 4KB physical memory pages. (This mapping is also used for external PEG devices.)

There is another logical mapping function available local to each graphics process; this works identically to the global GTT with some additional restrictions. It is providing each graphics context with its own local translation table and protected memory space (see Rendering Context Management later in this chapter).

The GTT and PPGTT are arrays of 4-byte *Page Table Entries* (PTEs) physically located in Main Memory. The GTT and PPGTT are comprised of a number of locked (non-swappable) physically-contiguous 4KB memory pages, with a maximum size (each) of 128 4KB pages (128K DWords map 128K*4KB = 512MB max) for Global GTT, and up to 512 4KB pages for PPGTT, for total up to 2GB max. GTT and PPGTT base addresses must be 4KB-aligned.

Note that the PTEs within the global GTT must be written only through GTTADDR (see the Device #2 Config registers for a description of this range), the invalidation of hardware TLBs need to be done explicitly with a fence prior to enabling the new surface using updated PTEs. [Reserved content]

The PPGTT base address is also 4KB aligned, but it is programmed directly in physical memory space rather than through an alias mechanism like GTTADDR. Note that not all clients may use the PPGTT; only the global GTT is available for processor accesses as well as graphics accesses from display engines (including overlay and cursor). Any per-process access that occurs while the PPGTT is disabled will default to a translation via the global GTT.

4.6.1 Changes to GTT

The GTT is constrained to be located at the beginning of a special section of stolen memory called the GTT stolen memory (GSM). There is no longer an MMIO register containing the physical base address of the GTT as on prior devices. Instead of using the PGTBL_CTL register to specify the base address of the GTT, the GTT base is now defined to be at the bottom (offset 0) of GSM. See the Device Cspecs for information on the location of GSM.

Since the graphics device (including the driver) knows nothing about the location of GSM, it does not “know” where the GTT is located in memory. In fact, the CPU cannot directly access the GSM containing the GTT. GTT entries therefore cannot be updated directly in memory as was possible on prior devices. Instead, the driver must use GTTADR as it could before, or it must make use of a new command (MI_UPDATE_GTT) to perform command-buffer-synchronized updates to the GTT.

Similarly, the Per-process GTT (PPGTT) is also constrained to be located within the GSM. The PPGTT base is specified by an offset in the PGTBL_CTL2 register. The PPGTT is treated as an extension of the GTT; it must be updated using either GTTADDR or the MI_UPDATE_GTT command.

The use by graphics of “physical” addressing can no longer be supported under VT-d. Therefore all graphics commands and registers that could, on prior devices, contain physical addresses must now use graphics memory (virtual) addresses. Functions that still require physical addressing are prohibited when VT-d is active. An example is hardware binning which requires physically addressed batch buffers. The hardware binner can only be used when VT-d is not active. Frame Buffer Compression (FBC) also requires physical addressing. FBC is not supported on Bearlake-C since it is a mobile feature and Bearlake-C is a desktop chipset.

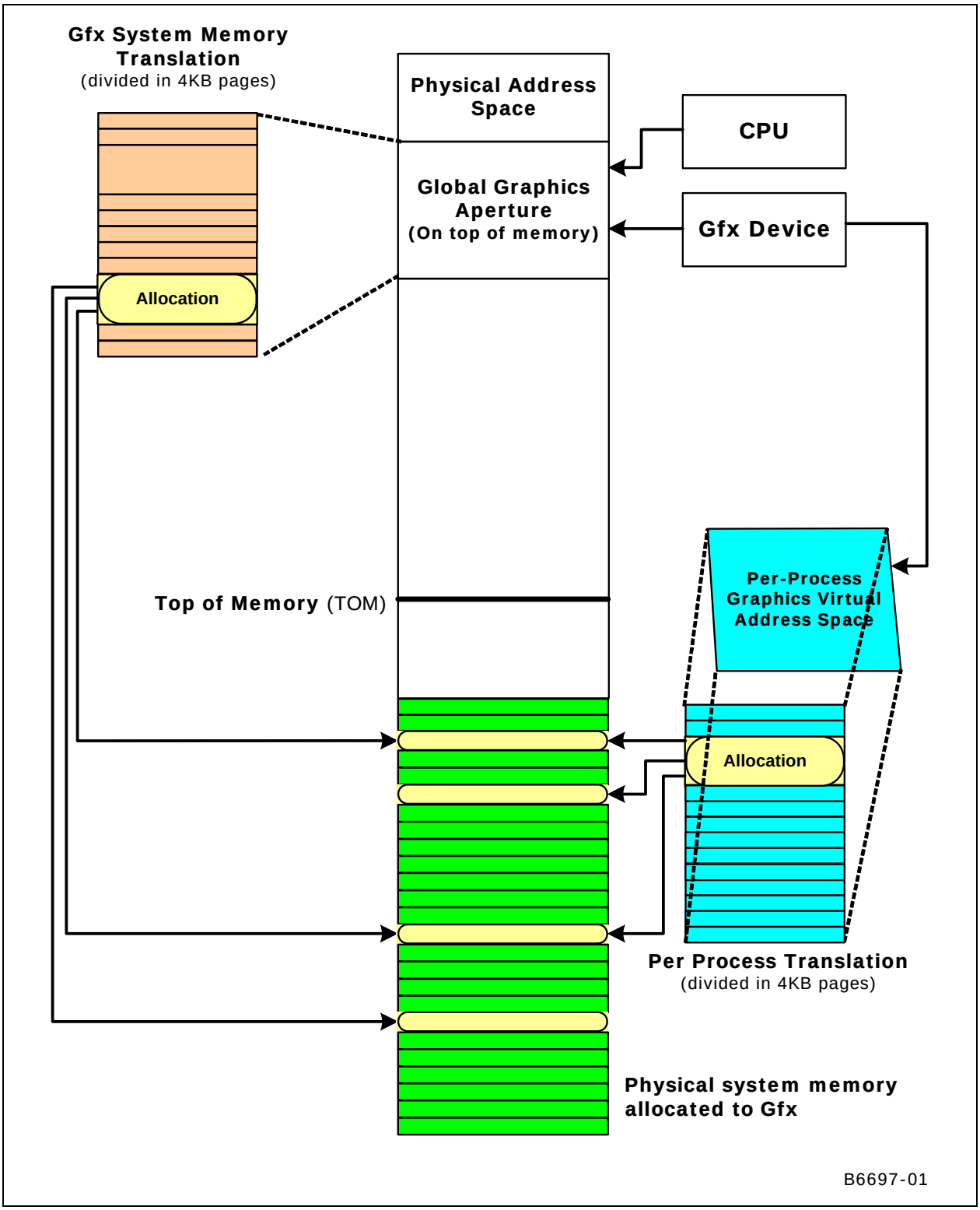
4.6.2 Logical Memory Space Mappings

Each valid PTE maps a 4KB page of Logical Memory to an independent 4KB page of:

- MM: Main Memory (unsnooped), or
- SM: System Memory (snooped, therefore coherent with the processor cache, must not be accessed through the Dev2_GM_Address range by the CPU)

PTEs marked as invalid have no backing physical memory, and therefore the corresponding Logical Memory Address pages must not be accessed in normal operation.

Figure 4-10. Global and Render GTT Mapping



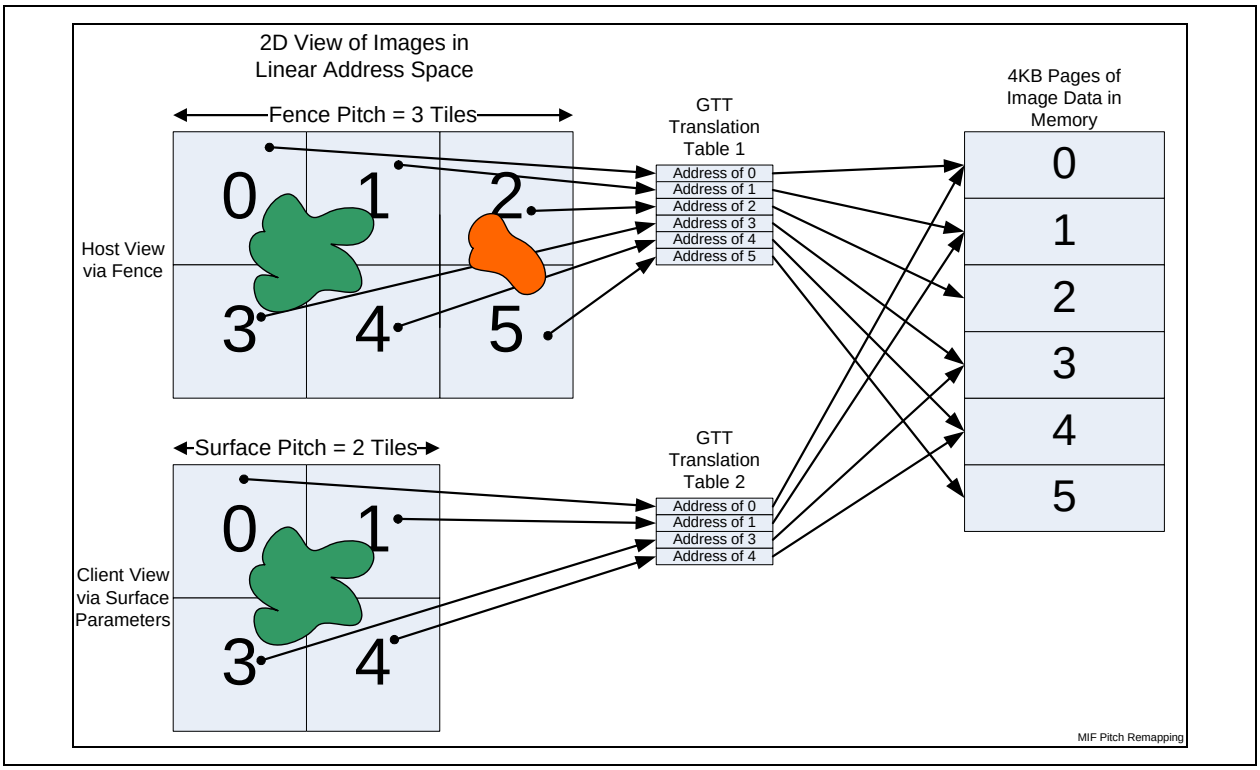
The following table lists the memory space mappings valid for each MI client:

MI Client	Logical Memory Space Mappings Supported	xGTT Usage
External Clients		
Host Processor	MM	GTT only
External PEG Device	None	n/a
Snooped Read/Write	None	n/a
Internal GPU Clients		
Render Command Ring Buffers	MM	GTT/PGTT, selected by PGTBL_STR2<2>
Render Command Batch Buffers	MM	GTT/PGTT, selected by PGTBL_STR2<5>
Indirect State Buffers	MM	GTT/PGTT, selected by PGTBL_STR2<4>
CURBE Constant Data	MM	Same xGTT used to fetch the CONSTANT_BUFFER command.
Media Object Indirect Data	MM	Same xGTT used to fetch the MEDIA_OBJECT command.
Vertex Fetch Data	MM, SM	GTT/PGTT, selected by PGTBL_STR2<3>
Sampler Cache (RO)	MM, SM	GTT/PGTT, selected by PGTBL_STR2<1>
DataPort Render Cache (R/W)	MM, SM	GTT/PGTT, selected by PGTBL_STR2<0>
Depth Buffer Cache (R/W)	MM	GTT/PGTT, selected by PGTBL_STR2<0>
Blit Engine	MM, SM	GTT/PGTT, selected by PGTBL_STR2<0>
MI_STORE_DATA_IMM Destination (if virtual addressed)	MM, SM	Same xGTT used to fetch the command.
PIPE_CONTROL Write Destination	MM, SM	GTT/PGTT, selected by the command
Display/Overlay Engines (internal)	MM	GTT only

Usage Note: Since the CPU cannot directly access memory pages mapped through a Graphics Process' local GTT (PPGTT), these pages must also be mapped though the global GTT (at least temporarily) in order for the CPU to initialize graphics data for a Graphics Process.

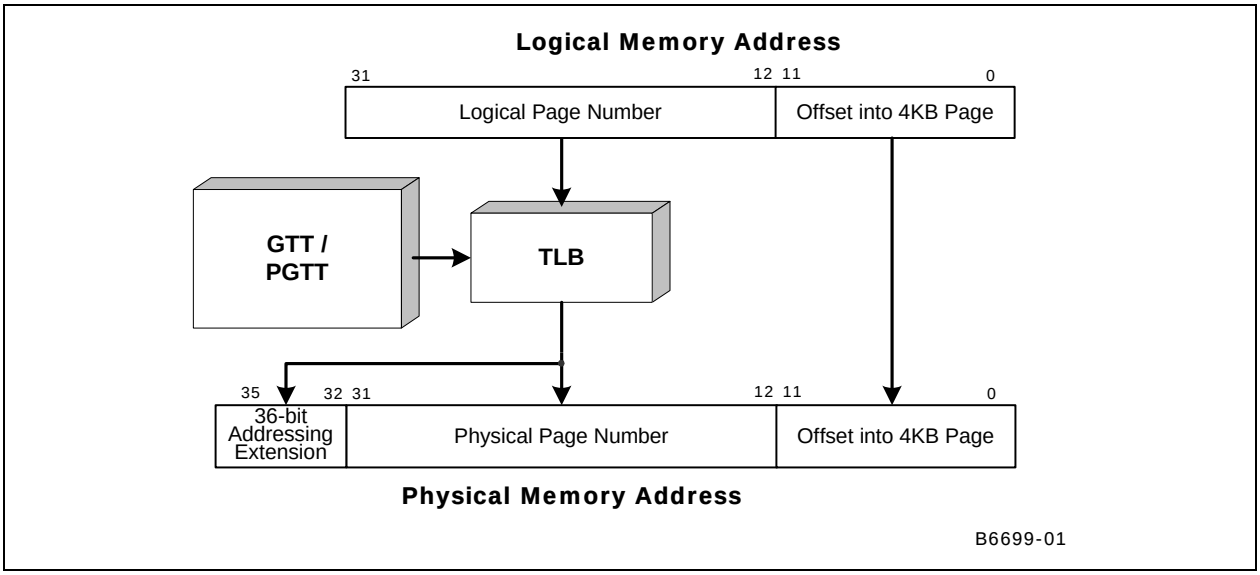
The PPGTT mechanism can be used by a client to access a surface with a pitch that is smaller than that of the fence region used by the host to initialize the surface, without having to physically move the data in memory.

Figure 4-11. GTT Re-mapping to Handle Differing Pitches



Refer to the "Graphics Translation Table (GTT) Range (GTTADR) & PTE Description" in *Memory Interface Registers* for details on PTE formats and programming information. Refer to the *Memory Data Formats* chapter for device-specific details/restrictions regarding the placement/storage of the various data objects used by the graphics device.

Figure 4-12. Logical-to-Physical Graphics Memory Mapping

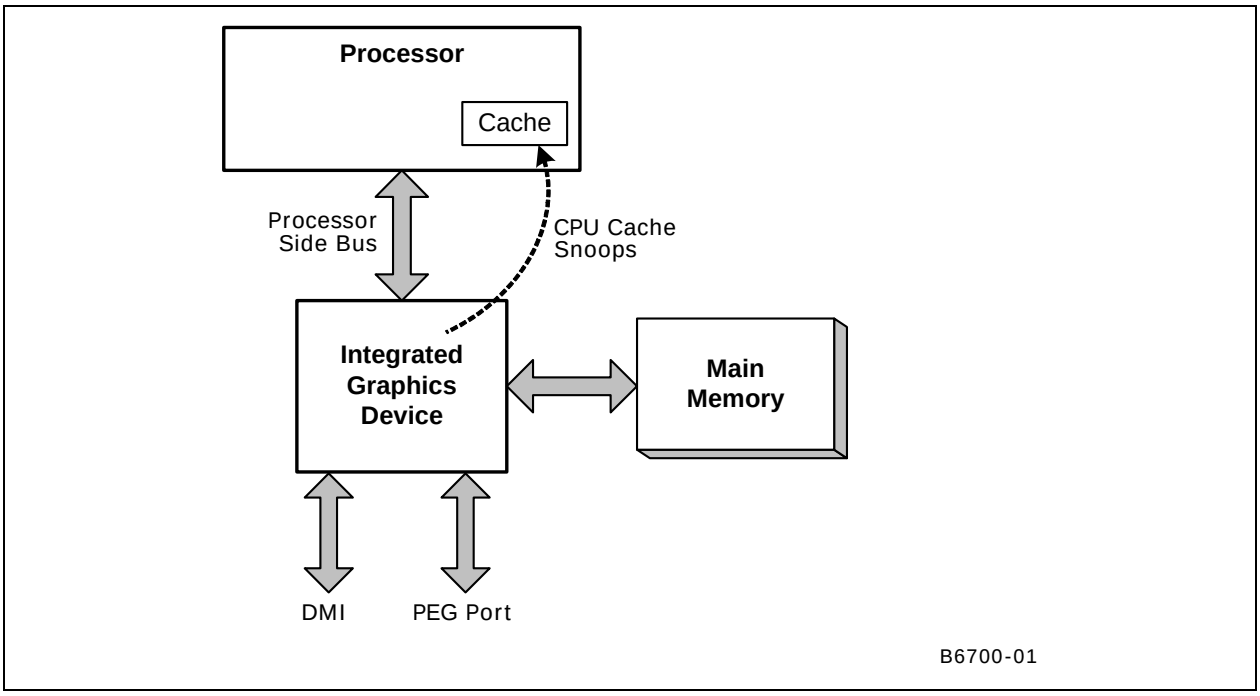


4.7 Physical Graphics Memory

The integrated graphics device satisfies all of its memory requirements using portions of main system memory. The integrated graphics device operates without any dedicated local memory, in a lower-cost configuration typically (though not necessarily officially) known as *Unified Graphics Memory* (UMA).

Figure 4-13 shows how the Main Memory is interfaced to the device.

Figure 4-13. Memory Interfaces



4.7.1 Physical Graphics Address Types

Table 4-4 lists the various physical address types supported by the integrated graphics device. Physical Graphics Addresses are either generated by Logical Memory mappings or are directly specified by graphics device functions. These physical addresses are not subject to tiling or GTT re-mappings.

Table 4-4. Physical Memory Address Types

Address Type	Description	Range
MM_Address	Main Memory Address. Offset into physical, unsnooped Main Memory.	[0,TopOfMemory-1]
SM_Address	System Memory Address. Accesses are snooped in processor cache, allowing shared graphics/ processor access to (locked) cacheable memory data.	[0,4GB]

4.7.2 Main Memory

The integrated graphics device is capable of using 4KB pages of physical main (system) memory for graphics functions. Some of this main memory can be “stolen” from the top of system memory during initialization (e.g., for a VGA buffer). However, most graphics operands are dynamically allocated to satisfy application demands. To this end the graphics driver will frequently need to allocate locked-down (i.e., non-swappable) physical system memory pages – typically from a cacheable non-paged pool. The locked pages required to back large surfaces are typically non-contiguous. Therefore a means to support “logically-contiguous” surfaces backed by discontiguous physical pages is required. The Graphics Translation Table (GTT) that was described in previous sections provides the means.

4.7.2.1 Optimizing Main Memory Allocation

This section includes information for software developers on how to allocate SDRAM Main Memory (SM) for optimal performance in certain configurations. The general idea is that these memories are divided into some number of page types, and careful arrangement of page types both within and between surfaces (e.g., between color and depth surfaces) will result in fewer page crossings and therefore yield somewhat higher performance.

The algorithm for allocating physical SDRAM Main Memory pages to logical graphics surfaces is somewhat complicated by (1) permutations of memory device technologies (which determine page sizes and therefore the number of pages per device row), (2) memory device row population options, and (3) limitations on the allocation of physical memory (as imposed by the OS).

However, the theory to optimize allocation by limiting page crossing penalties is simple: (a) switching between open pages is optimal (again, the pages do not need to be sequential), (b) switching between memory device rows does not in itself incur a penalty, and (c) switching between pages within a particular bank of a row incurs a page miss and should therefore be avoided.

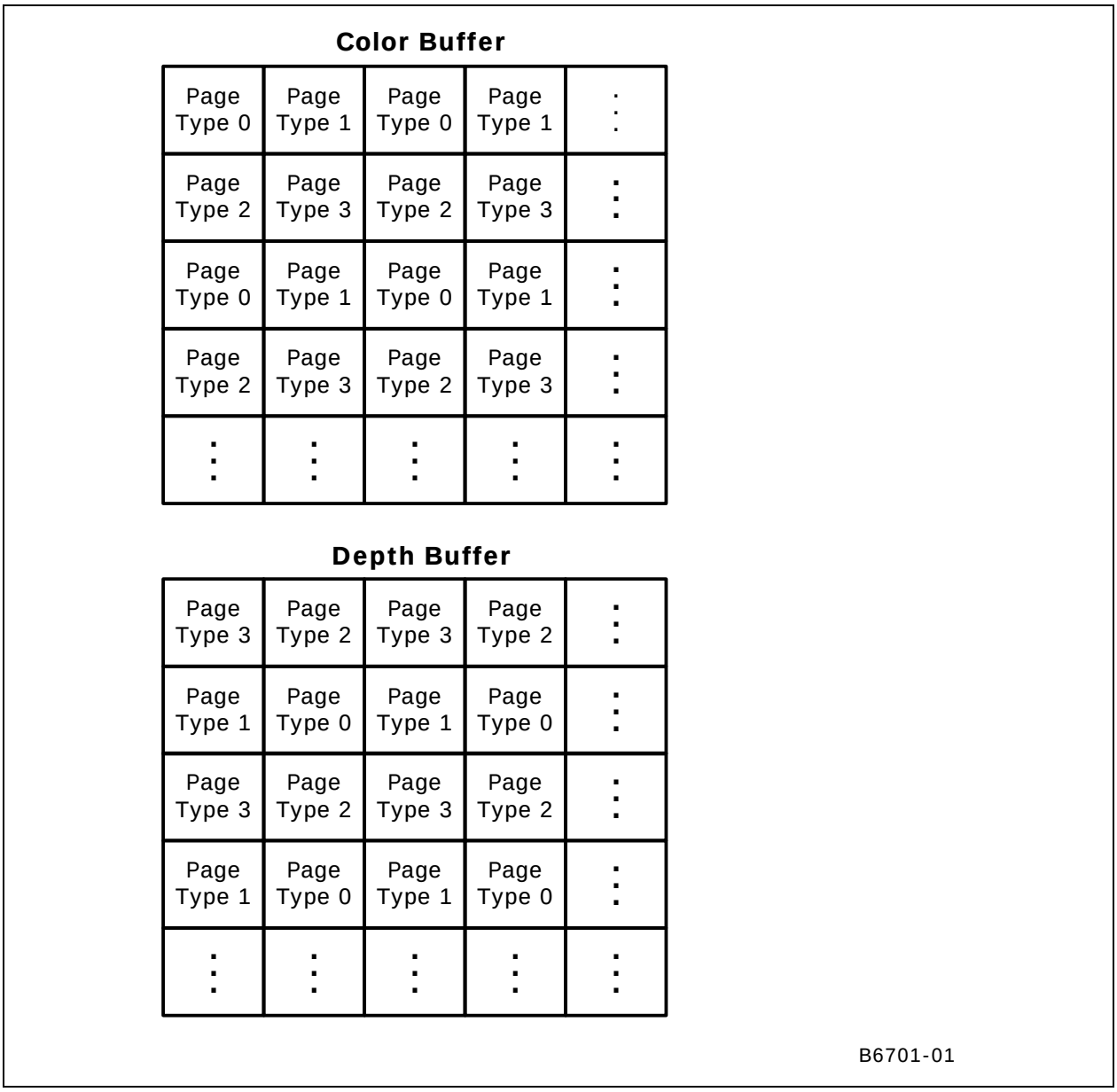
4.7.2.2 Application of the Theory (Page Coloring)

This section provides some scenarios of how Main Memory page allocation can be optimized.

4.7.2.2.1 3D Color and Depth Buffers

Here we want to minimize the impact of page crossings (a) between corresponding pages (1-4 tiles) in the Color and Depth buffers, and (b) when moving from a page to a neighboring page within a Color or Depth buffer. Therefore corresponding pages in the Color and Depth Buffers, and adjacent pages within a Color or Depth Buffer should be mapped to different page types (where a page’s “type” or “color” refers to the row and bank it’s in).

Figure 4-14. Memory Pages Backing Color and Depth Buffers



For higher performance, the Color and Depth Buffers could be allocated from different memory device rows.

4.7.2.2.2 Media/Video

The Y surfaces can be allocated using 4 page types in a similar fashion to the Color Buffer diagram above. The U and V surfaces would split the same 4 page types as used in the Y surface.

5. Device Programming Environment

The graphics device contains an extensive set of registers and commands (also referred to as "commands" or "packets") for controlling 2D, 3D, video I/O, and other operations. This chapter describes the programming environment and software interface to these registers/commands. The registers and commands themselves are described elsewhere in this document.

5.1 Programming Model

The graphics device is programmed via the following three basic mechanisms:

POST-Time Programming of Configuration Registers

These registers are the graphics device registers residing in PCI space. A majority of these registers are programmed once during POST of the video device. Configuration registers are not covered in this section.

Direct (Physical I/O and/or Memory-Mapped I/O) Access of Graphics Registers

Various graphics functions can only be controlled via direct register access. In addition, direct register access is required to initiate the (asynchronous) execution of graphics command streams. This programming mechanism is "direct" and synchronous with software execution on the CPU.

Command Stream DMA (via the Command Ring Buffer and Batch Buffers)

This programming mechanism utilizes the indirect and asynchronous execution of graphics command streams to control certain graphics functions, e.g., all 2D, 3D drawing operations. Software writes commands into a command buffer (either a Ring Buffer or Batch Buffer) and informs the graphics device (using the Direct method above) that the commands are ready for execution. The graphics device's Command Parser (CP) will then, or at some point in the future, read the commands from the buffer via DMA and execute them.

5.2 Graphics Device Register Programming

The graphics device registers (except for the Configuration registers) are memory mapped. The base address of this 512 KB memory block is programmed in the MMADR Configuration register. For a detailed description of the register map and register categories, refer to the *Register Maps* chapter.

Programming Note:

- Software must only access GR06, MSR0, MSR1, and Paging registers (see *Register Maps*) via Physical I/O, never via Memory Mapped I/O.

5.3 Graphics Device Command Streams

This section describes how command streams can be used to initiate and control graphics device operations.

5.3.1 Command Use

Memory-resident commands are used to control drawing engines and other graphics device functional units:

- **Memory Interface (MI) Commands**. The MI commands can be used to control and synchronize the command stream as well as perform various auxiliary functions (e.g., perform display/overlay flips, etc.)
- **2D Commands (BLT).** These commands are used to perform various 2D (Blt) operations.
- **3D Commands.** 3D commands are used to program the 3D pipeline state and perform 3D rendering operations. There are also a number of 3D commands that can be used to accelerate 2D and video operations, e.g., “StretchBlt” operations, 2D line drawing, etc.
- **Video (MPEG, etc.) Decode Commands**. A set of commands are supported to perform video decode acceleration including Motion Compensation operations via the Sampling Engine of the 3D pipeline.

5.3.2 Command Transport Overview

Commands are not written directly to the graphics device – instead they are placed in memory by software and later read via DMA by the graphics device’s Command Parser (CP) within the Memory Interface function. The primary mechanism used to transport commands is through the use of a Ring Buffer.

An additional, indirect mechanism for command transport is through the use of Batch Buffers initiated from the Ring buffer.

The Command Parser uses a set of rules to determine the order in which commands are executed. Following sections in this chapter provide descriptions of the Ring Buffer, Batch Buffers, and Command Parser arbitration rules.

5.3.3 Command Parser

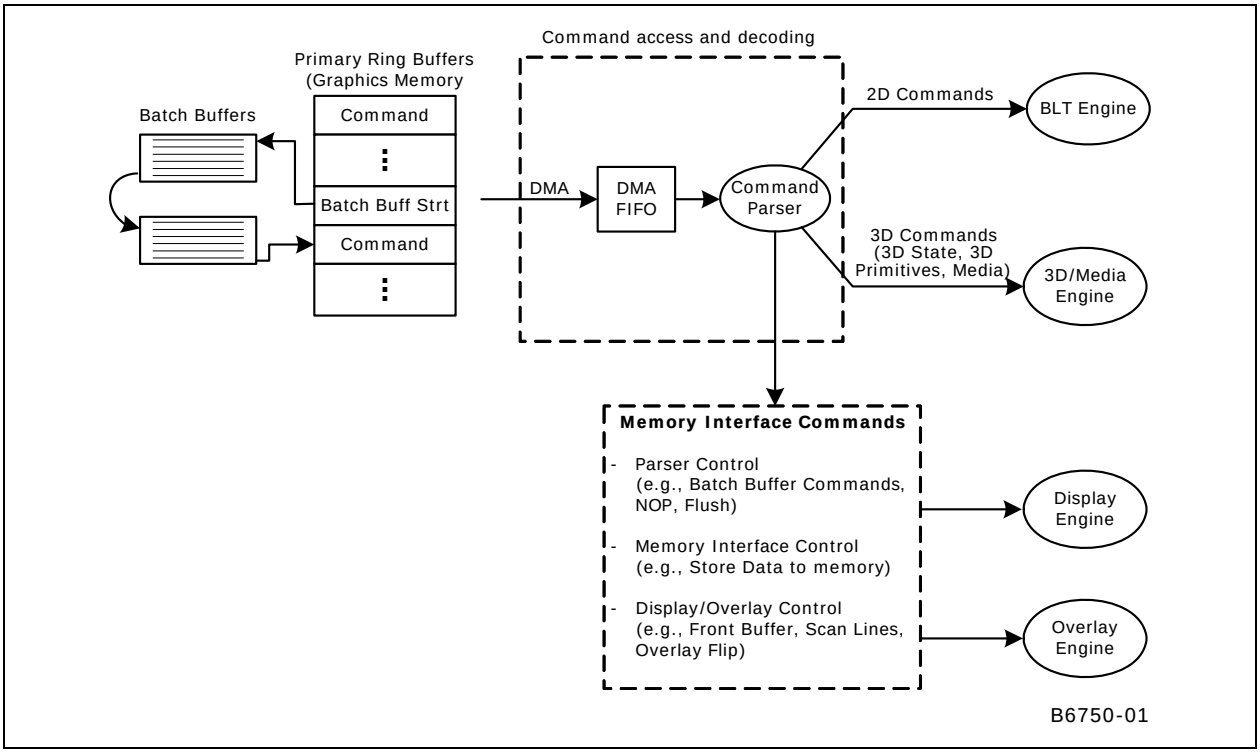
The graphics device’s Command Parser (CP) is responsible for:

- Detecting the presence of commands (within the Ring Buffer).
- Reading commands from the Ring Buffer and Batch Buffers via DMA. This includes support of the automatic head report function.
- Parsing the common "Command Type" (destination) field of commands.

- Execution of Memory Interface commands that control CP functionality, provide synchronization functions, and provide display and overlay flips as well as other miscellaneous control functions.
- Redirection of 2D, 3D and Media commands to the appropriate destination (as qualified by the INSTPM register) while enforcing drawing engine concurrency and coherency rules.
- Performing the “Sync Flush” mechanism
- Enforcing the Batch Buffer protection mechanism

Figure 5-1 is a high-level diagram of the graphics device command interface.

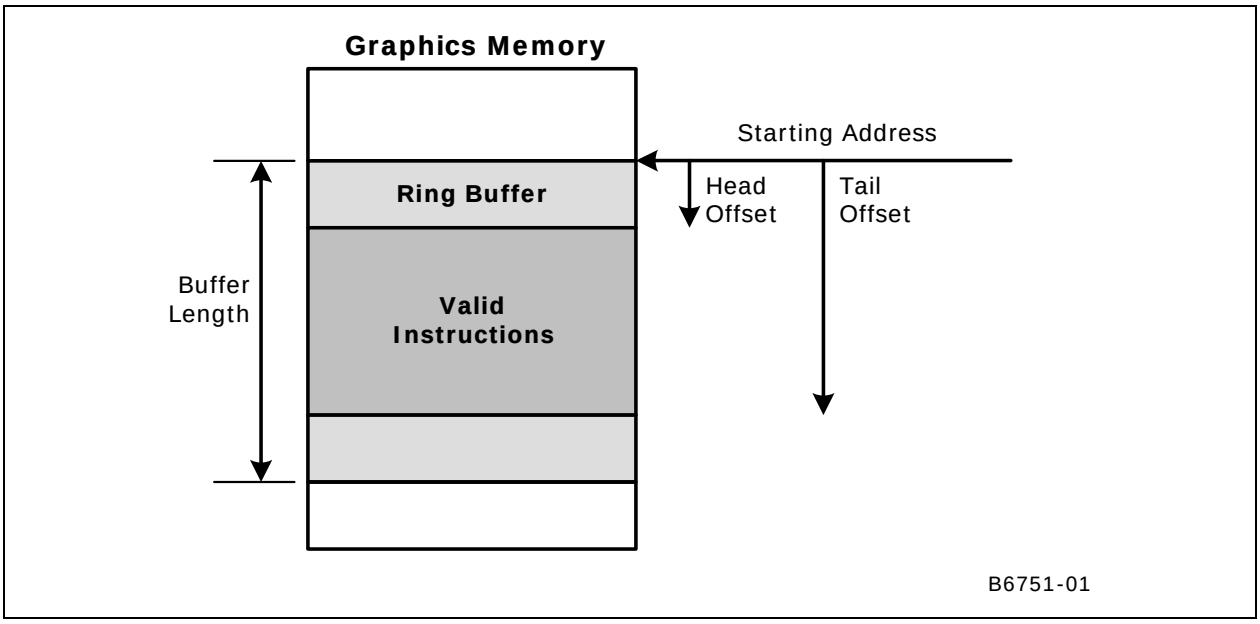
Figure 5-1. Graphics Controller Command Interface



5.3.4 The Ring Buffer

The ring buffesr are defined by a set of Ring Buffer registers and a memory area that is used to hold the actual commands. The Ring Buffer registers (described in full below) define the start and length of the memory area, and include two “offsets” (head and tail) into the memory area. Software uses the Tail Offset to inform the CP of the presence of valid commands that must be executed. The Head Offset is incremented by the CP as those commands are parsed and executed. The list of commands can wrap from the bottom of the buffer back to the top. Also included in the Ring Buffer registers are control fields that enable the ring and allow the head pointer to be reported to cacheable memory for more efficient flow control algorithms.

Figure 5-2. Ring Buffer



5.3.4.1 The Ring Buffer (RB)

Ring Buffer support:

- Batch Buffer initiation
- Indirect Data (operand access)

5.3.4.2 Ring Buffer Registers

A Ring Buffer is defined by a set of 4 Ring Buffer registers. Before a Ring Buffer can be used for command transport, software needs to program these registers. The fields contained within these registers are as follows:

- **Ring Buffer Valid:** This bit controls whether the Ring Buffer is included in the command arbitration process. Software must program all other Ring Buffer parameters before enabling a Ring Buffer. Although a Ring Buffer can be enabled in the non-empty state, it must not be disabled unless it is empty. Attempting to disable a Ring Buffer in the non-empty state is UNDEFINED. Enabling or disabling a Ring Buffer does not of itself change any other Ring Buffer register fields.
- **Start Address:** This field points to a contiguous, 4KB-aligned, linear (i.e., must not be tiled), mapped graphics memory region which provides the actual command buffer area.
- **Buffer Length:** The size of the buffer, in 4KB increments, up to 2MB.
- **Head Offset:** This is the DWord offset (from Start Address) of the next command that the CP will parse (i.e., it points one DWord past the last command parsed). The CP will update this field as commands are parsed – the CP typically continues parsing new commands before the previous

command operations complete. (Note that, if commands are pending execution, the CP will likely have prefetched commands past the Head Offset). As the graphics device does not "reset" the Head Offset when a Ring Buffer is enabled, software must program the Head Offset field before enabling the Ring Buffer. Software can enable a Ring Buffer with any legal values for Head/Tail (i.e., can enable the Ring Buffer in an non-empty state). It is anticipated, but not required, that software enable The Ring Buffer with Head and Tail Offsets of 0. Once the Head Offset reaches the QWord specified by the Tail Offset (i.e., the offsets are equal), the CP considers the Ring Buffer "empty".

- **Head Wrap Count:** This field is incremented by the CP every time the Head Offset wraps back to the start of the buffer. As it is included in the DWord written in the "report head" process, software can use this field to track CP progress as if the Ring Buffer had a "virtual" length of 2048 times the size of the actual physical buffer (up to 4GB).

- **Tail Offset:** This is the offset (from Start Address) of the next QWord of command data that software will request to be executed (i.e., it points one DWord past the last command DWord submitted for execution). The Tail Offset can only point to an command boundary – submitting partial commands is UNDEFINED. As the Tail Offset is a QWord offset, this requires software to submit commands in multiples of QWords (both DWords of the last QWord submitted must contain valid command data). Software may therefore need to insert a “pad” command to meet this restriction. After writing commands into the Ring Buffer, software updates the Tail Offset field in order to submit the commands for execution (by setting it to the QWord offset past the last command). The commands submitted can wrap from the end of the buffer back to the top, in which case the Tail Offset written will be less than the previous value. As the “empty” condition is defined as “Head Offset == Tail Offset”, the largest amount of data that can be submitted at any one time is one QWord less than the Ring Buffer length.

- **IN USE Semaphore Bit:** This bit (included in the Tail Pointer register) is used to provide a HW semaphore that SW can use to manage access to the individual The Ring Buffer. See the Ring Buffer Semaphore section below.

- **Automatic Report Head Enable:** Software can request to have the hardware Head Pointer register contents written ("reported") to snooped system memory on a periodic basis. Auto-reports can be programmed to occur whenever the Head Offset crosses either a 64KB or 128KB boundary. (Note therefore that a Ring Buffer must be at least 64KB in length for the auto-report mechanism to be useful). The complete Head Pointer register will be stored at a Ring Buffer-specific DWord offset into the "hardware status page" (defined by the HWSTAM register). The auto-report mechanism is desirable as software needs to use the Head Offset to determine the amount of free space in the Ring Buffer -- and having the Head Pointer periodically reported to system memory provides a fairly up-to-date Head Offset value automatically (i.e., without having to explicitly store a Head Pointer value via the MI_REPORT_HEAD command).

Table 5-1. Ring Buffer Characteristics

Characteristic	Description
Alignment	4 KB page aligned.
Max Size	2 MB
Length	Programmable in numbers of 4 KB pages.
Start Pointer	Programmable 4KB page-aligned address of the buffer
Head pointer	Hardware maintained DWord Offset into the ring buffer. Commands can wrap. Programmable to initially set up ring.
Tail pointer	Programmable QWord Offset into the ring buffer – indicating the *next* QWord where software can insert new commands.

5.3.4.3 Ring Buffer Placement

Ring Buffer memory buffers are defined via a Graphics Address and must physically reside in (uncached) Main Memory. There is no support for The Ring Buffer in cacheable system memory.

5.3.4.4 Ring Buffer Initialization

Before initializing a Ring Buffer, software must first allocate the desired number of 4KB pages for use as buffer space. Then the Ring Buffer registers associated with the Ring Buffer can be programmed. Once the Ring Buffer Valid bit is set, the Ring Buffer will be considered for command arbitration, and the Head and Tail Offsets will either indicate an empty Ring Buffer (i.e., Head Offset == Tail Offset), or will define some amount of command data to be executed.

5.3.4.5 Ring Buffer Use

Software can write new commands into the "free space" of the Ring Buffer, starting at the Tail Offset QWord and up to the QWord prior to the QWord indicated by the Head Offset. Note that this "free space" may wrap from the end of the Ring Buffer back to the start (hence the “ring” in the name).

While the “free space” wrap may allow commands to be wrapped around the end of the Ring Buffer, the wrap should only occur between commands. Padding (with NOP) may be required to follow this restriction.

Software is required to use some mechanism to track command parsing progress in order to determine the "free space" in the Ring Buffer. This can be accomplished in one of the following ways:

1. A direct read (poll) of the Head Pointer register. This gives the most accurate indication but is expensive due to the uncached read.

2. The automatic reporting of the Head Pointer register in the Hardware Status Page. This has low impact as no uncached reads or command overhead is involved. However, given the 64KB/128KB granularity of auto-reports, this mechanism only works well on fairly large The Ring Buffer.

3. The explicit reporting of the Head Pointer register via the MI_REPORT_HEAD command. This allows for flexible and more accurate reporting but comes at the cost of command bandwidth and execution time, in addition to the software overhead to determine how often to report the head.

4. Some other "implicit" means by which software can determine how far the CP has progressed in retiring commands from a Ring Buffer. This could include the use of "Store DWORD" commands to write sequencing data to system memory. This has similar characteristics to using the MI_REPORT_HEAD mechanism.

Once the commands have been written and, if necessary, padded out to a QWord, software can write the Tail Pointer register to submit the new commands for execution. The uncached write of the Tail Pointer register will ensure that any pending command writes are flushed from the processor.

If the Ring Buffer Head Pointer and the Tail Pointer are on the same cacheline, the Head Pointer must not be greater than the Tail Pointer.

5.3.4.6 Ring Buffer Semaphore

When the **Ring Buffer Mutex Enable** (RBME) bit if the INSTPM MI register is clear, all Tail Pointer IN USE bits are disabled (read as zero, writes ignored). When RBME is enabled, the IN USE bit acts as a Ring Buffer semaphore. If the Tail Pointer is read, and IN USE is clear, it is immediately set after the read. Subsequent Tail Pointer reads will return a set IN USE bit, until IN USE is cleared by a Tail Pointer write.

This allows SW to maintain exclusive ring access through the following protocol: A SW agent needing exclusive ring access must read the Tail Pointer before accessing the Ring Buffer: if the IN USE bit is clear, the agent gains access to the Ring Buffer; if the IN USE bit is set, the agent has to wait for access to the Ring Buffer (as some other agent has control). The mechanism to inform pending agents upon release of the IN USE semaphore is unspecified (i.e., left up to software).

5.3.5 Batch Buffers

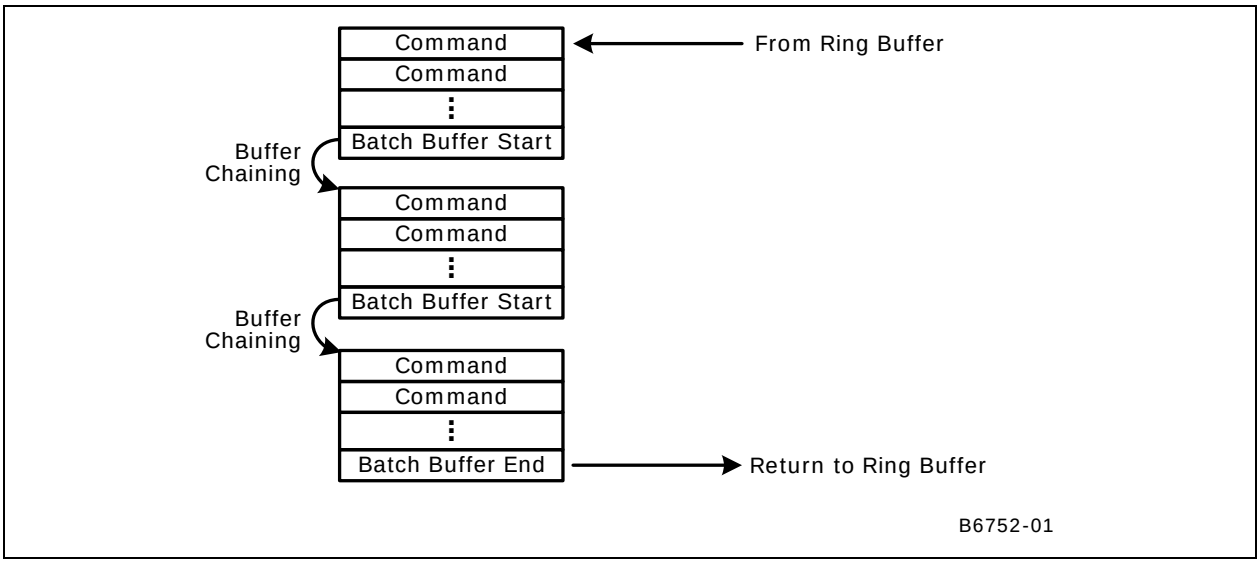
The graphics device provides for the execution of command sequences *external* to the Ring buffer. These sequences are called "Batch Buffers", and are initiated through the use of various Batch Buffer commands described below. When a Batch Buffer command is executed, a batch buffer sequence is initiated, where the graphics device fetches and executes the commands sequentially via DMA from the batch buffer memory.

5.3.5.1 Batch Buffer Chaining

What happens when the end of the Batch Buffer is reached depends on the final command in the buffer. Normally, when a Batch Buffer is initiated from a Ring Buffer, the completion of the Batch Buffer will cause control to pass back to the Ring Buffer at the command following the initiating Batch Buffer command.

However, the final command of a Batch Buffer can be another Batch Buffer-initiating command (MI_BATCH_BUFFER_START). In this case control will pass to the new Batch Buffer. This process, called *chaining*, can continue indefinitely, terminating with a Batch Buffer that does not chain to another Batch Buffer (ends with MI_BATCH_BUFFER_END) – at which point control will return to the Ring Buffer.

Figure 5-3. Batch Buffer Chaining



5.3.5.2 Ending Batch Buffers

The end of the Batch Buffer is determined as the buffer is being executed: either by (a) an MI_BATCH_BUFFER_END command, or (b) a "chaining" MI_BATCH_BUFFER_START command. There is no explicit limit on the size of a Batch Buffer that uses GTT-mapped memory. Batch buffers in physical space cannot exceed one physical page (4KB).

5.3.6 Indirect Data

In addition to Ring Buffer and Batch Buffers, the MI supports the access of *indirect* data for some specific command types. (Normal read/write access to surfaces isn't considered indirect access for this discussion).

5.3.6.1 Logical Contexts

Logical Contexts, indirectly referenced via the MI_SET_CONTEXT command, must reside in (unsnooped) Main Memory.

5.3.7 Command Arbitration

The command parser employs a set of rules to arbitrate among these command stream sources. This section describes these rules and discusses the reasoning behind the algorithm.

5.3.7.1 Arbitration Policies and Rationale

The Ring buffer (RB) is considered the primary mechanism by which drivers will pass commands to the graphics device.

The insertion of command sequences into the Ring Buffer must be a "synchronous" operation, i.e., software must guarantee mutually exclusive access to the Ring Buffer among contending sources (drivers). This ensures that one driver does not corrupt another driver's partially-completed command stream. There is currently no support for unsynchronized multi-threaded insertion of commands into ring buffer.

Another requirement for asynchronous command generation arises from competing (and asynchronous) drivers (e.g., "user-mode" driver libraries). In this case, the desire is to allow these entities to construct command sequences in an asynchronous fashion, via batch buffers. Synchronization is then only required to "dispatch" the batch buffers via insertion of Batch Buffer commands inserted into the Ring Buffer.

Software retains some control over this arbitration process. The MI_ARB_ON_OFF command disables all other sources of command arbitration until re-enabled by a subsequent MI_ARB_ON_OFF command from the same command stream. This can be used to define uninterruptible "critical sections" in an command stream (e.g., where some device operation needs to be protected from interruption). Disabling arbitration from a batch buffer without re-enabling before the batch is complete is UNDEFINED.

Batch Buffers can be (a) interruptible at command boundaries, (b) interruptible only at chain points, or (c) non-interruptible. See MI_BATCH_BUFFER_START in *Memory Interface Commands* for programming details.

5.3.7.2 Wait Commands

The MI_WAIT_EVENT command is provided to allow command streams to be held pending until an asynchronous event occurs or condition exists. An *event* is defined as occurring at a specific point in time (e.g., the leading edge of a signal, etc.) while a *condition* is defined as a finite period of time. A wait on an event will (for all intents and purposes) take some non-zero period of time before the subsequent command can be executed. A wait on a condition is effectively a noop if the condition exists when the MI_WAIT_EVENT command is executed.

A Wait in the Ring Buffer or batch buffer will cause the CP to treat the Ring Buffer as if it were empty until the specific event/condition occurs. This will temporarily stall the Ring Buffer.

While the Ring Buffer is waiting, the **RB Wait** bit of the corresponding RB*n*_CTL register will be set. Software can cancel the wait by clearing this bit (along with setting the **RB Wait Write Enable** bit). This will terminate the wait condition and the Ring Buffer will be re-enabled. This sequence can be included when software is required to flush all pending device operations and pending Ring Buffer waits cannot be tolerated.

5.3.7.3 Wait Events/Conditions

This section describes the wait events and conditions supported by the MI_WAIT_EVENT command. Only one event or condition can be specified in an MI_WAIT_EVENT, though different command streams can be simultaneously waiting on different events.

5.3.7.3.1 Display Pipe A,B Vertical Blank Event

The Vertical Blank event is defined as “shortly after” the *leading edge* of the next display VBLANK period of the corresponding display pipe. The delay from the leading edge is provided to allow for internal device operations to complete (including the update of display and overlay status bits, and the update of overlay registers).

5.3.7.3.2 Display Pipe A,B Horizontal Blank Event

The Horizontal Blank event is defined as “shortly after” the *leading edge* of the next display HBLANK period of the corresponding display pipe.

5.3.7.3.3 Display Plane A, B, Sprite A, Sprite B Flip Pending Condition

The Display Flip Pending condition is defined as the period starting with the execution of a “flip” (MI_DISPLAY_BUFFER_INFO) command and ending with the completion of that flip request. Note that the MI_DISPLAY_BUFFER_INFO command can specify whether the flip should be synchronized to vertical refresh or completed “as soon as possible” (likely some number of horizontal refresh cycles later).

5.3.7.3.4 Display Pipe A,B Scan Line Event

The Scan Line Event is defined as the start of a scan line specified in the Pipe A Display Scan Line Count Range Compare Register.

5.3.7.3.5 Semaphore Wait Condition

One of the 8 defined condition codes contained within the Execute Condition Code (EXCC) Register can be selected as the source of a wait condition. While the selected condition code bit is set, the initiating command stream will be removed from arbitration (i.e., paused). Arbitration of that command stream will resume once the condition code bit is clear. If the selected condition code is clear when the WAIT_ON_EVENT is executed, the command is effectively ignored.

5.3.7.4 Command Arbitration Points

The CP performs arbitration for command execution at the following points:

- Upon execution of an MI_ARB_CHECK command
- When the ring buffer becomes empty

5.3.7.5 Command Arbitration Rules

At an arbitration point, the CP will switch to the new head pointer contained in the UHPTR register if it is valid. Otherwise it will idle if empty, or continue execution in the current command flow if it arbitrated due to an MI_ARB_CHECK command.

5.3.7.6 Batch Buffer Protection

The CP employs a protection mechanism to help prevent random writes to system memory from occurring as a result of the execution of a batch buffer generated by a “non-secure” agent (e.g., client-mode library). Commands executed directly from a ring buffer, along with batch buffers initiated from a ring buffer and marked as “secure”, will not be subject to this protection mechanism as it is assumed they can only be generated by “secure” driver components.

This protection mechanism is enabled via a field in a Batch Buffer command that indicates whether the associated batch buffer is “secure” or “non-secure”. When the CP processes a non-secure batch buffer from the ring buffer it does not allow any MI_STORE_DATA_IMM commands that reference physical addresses, as that would allow the non-secure source to perform writes to any random DWord in the system. (Note that graphics engines will only write to graphics memory ranges, which by definition are virtual memory ranges mapped into physical memory pages by trusted driver components using the GTT/TGTT hardware). Placing an MI_STORE_DATA in a non-secure batch buffer will instead cause a Command Error. The CP will store the header of the command, the origin of the command, and an error code. In addition, such a Command Error can generate an interrupt or a hardware write to system memory (if these actions are enabled and unmasked in the IER and IMR registers respectively.) At this point the CP can be reactivated only by a **full reset**.

The security indication field of Batch Buffer instructions placed in batch buffers (i.e., “chaining” batch buffers) is ignored and the chained batch buffer will therefore inherit the security indication of the first Batch Buffer in the chain (i.e. the batch buffer that was initiated by an MI_BATCH_BUFFER_START command in the Ring Buffer).

5.3.8 Graphics Engine Synchronization

This table lists the cases where engine synchronization is required, and whether software needs to ensure synchronization with an explicit MI_FLUSH command or whether the device performs an implicit (automatic) flush instead. Note that a pipeline flush can be performed without flushing the render cache, but not vice versa.

Event	Implicit Flush or Requires Explicit Flush?
PIPELINE_SELECT	Requires explicit pipeline flush
Any Non-pipelined State Command	Device implicitly stalls the command until the pipeline has drained sufficiently to allow the state update to be performed without corrupting work-in-progress
MI_SET_CONTEXT	Device performs implicit flush
MI_DISPLAY_BUFFER_INFO (“display flip”)	Requires explicit render cache flush
MI_OVERLAY_FLIP	Requires explicit render cache flush
3D color destination buffer (render target) used as texture (i.e., “rendered texture”)	Requires explicit render cache flush
MEDIA_STATE_POINTERS	Requires explicit pipeline flush
MEDIA_OBJECT	Requires explicit pipeline flush
Media: Previous Destination Used as Source	Requires explicit render cache flush

Event	Implicit Flush or Requires Explicit Flush?
MI_SEMAPHORE_MBOX update to another pipe that requires output of current pipe to be in coherent memory. Example is blitter/render pipe synchronization	Requires explicit pipe flush

For GEN6, any time a pipeline flush is used it will need to follow by 8 separate storeDW commands. This is done as a work-around to a known issue where the pipeline flush does not push all cycles to the "GO" point. The storeDWs will not accomplish this process but will create enough stall time to ensure all cycles are at GO prior to execution of the next buffer or part of the buffer. This is required for producer/consumer model.

5.3.9 Graphics Memory Coherency

Table 5-2 lists the various types of graphics memory coherency provided by the device, specifically where the CPU writes to a 64B cacheline, and the device then accesses that same cacheline. Note that the coherency policy depends on the address type (GM or MM) involved in the accesses.

Table 5-2. Graphics Memory Coherency

CPU Access	Subsequent Device Access	Example Operand	Coherency
Write GM	Read GM	Texture loads	GM accesses from IA cores use WC memory type, and any tail-pointer update is US, hence flow is coherent when device accesses GM.
Write MM	Read MM	Batch Buffer	IA writes are always coherent with device accesses, hardware ensures the coherency.
Write GM	Write GM		Device ensures coherency following every Ring Buffer Tail Pointer write. (This can be made optional via a bit in the Tail Pointer data).
Write MM	Write MM		"assumed to exclusive byte" No byte sharing w/o semaphores
Write GM	Read MM		Device ensures coherency following every Ring Buffer Tail Pointer write. (This can be made optional via a bit in the Tail Pointer data).

5.3.10 Graphics Cache Coherency

There are several caches employed within the graphics device implementation. This section describes the impact of these caches on the programming model (i.e., if/when does software need to be concerned).

5.3.10.1 Rendering Cache

The rendering (frame buffer) cache is used by the blit and 3D rendering engines and caches portions of the frame buffer color and depth buffers. This cache is guaranteed to be flushed under the following conditions (note that the implementation may flush the cache under additional, implementation-specific conditions):

- Execution of an MI_FLUSH command with the **Render Flush Cache Inhibit** bit clear
- Execution of a PIPE_CONTROL instruction with the **Write Cache Flush Enable** bit set (Depth Stall must be clear).
- A SyncFlush handshake operation
- A change of rendering engines (e.g., going from 2D to 3D, 3D to 2D, etc.)
- Logical Context switch (via MI_SET_CONTEXT)

The render cache must be explicitly flushed using one of these mechanisms under certain conditions. See Graphics Engine Synchronization above.

5.3.10.2 Sampler Cache

The read-only sampler cache is used to cache texels and other data read by the Sampling Engine in the 3D pipeline. This cache can be enabled or disabled via the **Texture L2 Disable** bit of the Cache_Mode_0 register (see *Memory Interface Registers*). Note that, although there may be more than one level of sampler cache within the implementation, the sampler cache is exposed as a single entity at the programming interface.

The sampler cache is guaranteed to be invalidated under the following conditions (note that the implementation may invalidate the cache under additional, implementation-specific conditions):

- Execution of an MI_FLUSH command with the **Map Cache Invalidate** bit set
- Execution of PIPE_CONTROL with the **Depth Stall Enable** bit clear.
- A SyncFlush handshake operation

The sampler cache must be invalidated prior to reallocation of physical texture memory (i.e., software must guarantee that stale texture data is invalidated before reusing physical texture memory for a new or modified texture).

5.3.10.3 Instruction/State Cache

The read-only ISC is used to cache pipelined state and EU instructions read in from memory. It also functions as a prefetch cache by reading in additional state information and instructions beyond those immediately requested in order to decrease latency and improve performance. As with the sampler cache, there may be more than one level of ISC within the implementation. The ISC is exposed as a single entity at the programming interface.

The instruction/state cache is guaranteed to be invalidated under the following conditions (note that the implementation may invalidate the cache under additional, implementation-specific conditions):

- Execution of an MI_FLUSH command with the **State/Instruction Cache Invalidate** bit set
- Execution of PIPE_CONTROL with the **Instruction/State Cache Flush Enable** bit set.
- A SyncFlush handshake operation

The instruction/state cache must be invalidated prior to reallocation of physical state/instruction memory (i.e., software must guarantee that stale state/instruction data is invalidated before reusing physical state/instruction memory for new or modified state or instructions).

5.3.10.4 Vertex Cache

The vertex cache consists of 2 sub-caches: one that caches vertex buffer data based on address, and another that caches (possibly shaded) vertex attribute data based on index (see the *Vertex Fetch* chapter for vertex index details). The latter cache is always invalidated between primitive topologies.

Both vertex caches are guaranteed to be invalidated under the following conditions (note that the implementation may invalidate the cache under additional, implementation-specific conditions):

- Execution of an MI_FLUSH command
- Execution of a PIPE_CONTROL command
- A SyncFlush handshake operation
- Logical Context switch (via MI_SET_CONTEXT)

5.3.10.5 GTT TLBs

The following table summarizes when the various TLBs are invalidated.

TLB	Normal Invalidation Mechanism
Display	Refreshed on Vsync
Overlay	Refreshed on Vsync
Render/Blit	Internal Flush*
Host	Through a Page Table PTE write
Sampler Cache	Internal Flush*
Command Stream	Through a Page Table PTE write

* -- Includes MI_FLUSH, Engine switch, and Context switch.

On [DevCTG], all TLBs are invalidated on a VT-d translation enable (TE) status change.

5.3.11 Command Synchronization

This section describes the hardware mechanisms that can be used by software to provide synchronization with command stream parsing and execution.

The key point here is distinguishing between command *parsing* and *retirement* – in that, for most commands, there is some finite delay between the parsing of a command and the retirement (coherent completion) of the operation it specifies.

Interrogation of the Ring Buffer Head Pointer only gives an indication of the progress of command parsing. This information is required to discern the availability of command data within the Ring Buffer or Batch Buffers. If the Head Pointer indicates the command data has been parsed, those locations can be reused, otherwise the commands must be considered still pending parsing and left alone.

Given the CP rules for command execution, it is possible to use the indication of command parsing progress to infer the retirement status of parsed commands. The only indication of instruction retirement available from instruction parsing is that parsing of an MI instruction implies retirement of previous MI instructions with the following exceptions:

- The parsing of a Memory Interface (MI) command implies that all previously-parsed MI commands have completed, with the following exceptions:
    - Display and Overlay Flip commands: Only the submission of the flip request is guaranteed. The flip operation will occur some time later. Mechanisms to detect the actual completion of a flip operation are described below.
    - "Store-Data" type commands: Only the submission of the store operation is guaranteed. The write result will be complete (coherent) some time later (this is practically a finite period but there is no guaranteed latency).
    - Batch Buffer commands: There is no guarantee that the operations performed by the batch buffer have completed.

Other than the cases described above, additional measures must be taken to discern the progress of command retirement. These measures are described in the following subsections.

5.3.11.1 MI_FLUSH

The MI_FLUSH command pauses further command parsing until all drawing engines become idle and any internal rendering cache is flushed and invalidated. All previous rendering commands can therefore be considered retired.

This flush operation is considered complete once command parsing proceeds to the next command. Software can, for example, follow an MI_FLUSH command with an MI_STORE_DATA_IMM or MI_STORE_DATA_INDEX command – where the completion of the store operation implies that the flush operation has completed. (Note that if the last DWord in a ring buffer is an MI_FLUSH instruction, there is no way by simply looking at the Ring Buffer registers to determine whether the flush operation is complete or still pending.)

The successful completion of an MI_FLUSH command does not guarantee that *all* previous operations have completed. Operations that may still be pending include:

- Store Data type commands (MI_STORE_DATA_IMM, MI_STORE_DATA_INDEX, MI_REPORT_HEAD)
- Display or Overlay Flip operations

See section 5.3.10.2 for more information on when the sampler cache should be invalidated.

5.3.11.2 Sync Flush

Inserting MI_FLUSH commands, while effective at determining or forcing the retirement of previous rendering commands, may negatively impact performance if not absolutely required. For example, if the knowledge of rendering command retirement is not known a priori, it is likely undesirable to insert MI_FLUSH commands at intervals in the command stream. However, it may not be acceptable to insert an MI_FLUSH command (and wait for its completion) at the point that rendering command retirement is required – as there may be a large number of commands pending in ring/batch buffers at that point and flushing the entire device (including waiting for completion of pending commands that have not yet been parsed) may be prohibitive. There is a mechanism, however, where command stream synchronization can be performed on demand, without requiring earlier submitted commands and batch buffers to complete – it is called the "Sync Flush" mechanism.

Here's how it works:

- Software must (preferably at driver initialization time) unmask the Sync Status bit in the Hardware Status Mask Register (HWSTAM). This should be done unconditionally (at least whenever HW status writes are enabled), as any bandwidth increase due to Sync Status-initiated writes is negligible.
- At the point that synchronization is required, software must guarantee that command parsing has progressed past the point of interest in the command stream (i.e., past the last command whose retirement is required). Note that this step is required in any scheme.
- Software then reads the location where the Interrupt Status is reported in the Hardware Status Page (DWord offset 0) and saves that DWord in a temporary variable.
- Software then sets the Sync Enable bit of the Command Parser Mode Register (INSTPM) via an uncached write.
- The Command Parser will detect the Sync Enable bit set before it proceeds to the very next command (or immediately if the CP is idle). It will then perform an internal flush operation. This flush is identical to that performed by an MI_FLUSH command with all flush types enabled.
- Once this flush operation is complete, the CP will clear the Sync Enable bit of the INSTPM register and then *toggle* the Sync Status bit of the ISR register. This will initiate a write of the ISR register contents (with the toggled Sync Status) to DWord 0 of the Hardware Status page (as part the normal hardware status write mechanism).
- Software, following the write of the INSTPM register, should periodically poll the Hardware Status location. By comparing the current versus saved value of the Sync Status bit, software can then detect when the flush operation is complete. Note that the latency of this operation is typically small, as it will be initiated either immediately or at least before the next command is parsed (regardless of arbitration conditions).

5.4 Hardware Status

The graphics device supports a number of internal hardware status bits which can be used to detect and monitor hardware status conditions via polling or interrupts. This section will describe each hardware status bit. The following section describes the hardware status reporting (polling) mechanism. The mechanism to allow these status bits to generate interrupts is described in the Interrupts section. Note that the hardware status bits are actually reported in the Interrupt Status Register, so "hardware status"

and “interrupt status” are used interchangeably here (though many hardware status bits won’t necessarily ever be used to generate interrupts).

The following subsections describe the various hardware (interrupt) status bits, as defined in the Interrupt Status Register.

5.4.1 Hardware-Detected Errors (Master Error bit)

This interrupt status bit is generated whenever an “unmasked” hardware-detected error status is detected. See the Errors section.

5.4.2 Thermal Sensor Event

This interrupt status bit is generated by “thermal events” detected by the Thermal Sensor logic. The bit corresponding to this event in the HWSTAM register must always be masked (i.e., set to ‘1’) so that thermal sensor events do not generate HW status DWord writes. See Hardware Status Writes.

5.4.3 Sync Status

This bit should only be used as described in Sync Flush, and should not be used to generate interrupts (i.e., the corresponding interrupt should not be enabled in the IER).

5.4.4 Display Plane A, B, (Sprite A, Sprite B) Flip Pending

These bits are used to report the status of “flip” operations on the corresponding Display Plane. Display Flip operations are requested via the MI_DISPLAY_BUFFER_INFO command. When that command is executed, the corresponding Display Flip Pending status in the ISR register will be set to ‘1’ indicating that a display flip has been requested but has not yet been performed. (Requesting a flip operation when one is already pending is UNDEFINED). This indicates that a flip is “pending”. At the appropriate time during the next vertical blank period (for that display stream), the flip operation will be performed (i.e., the display will switch to refreshing from the new display buffer). This causes the Display Flip Pending status to reset to ‘0’. When this occurs, and the Display Flip Pending status bit is unmasked by the Interrupt Mask Register (IMR), the Display Flip Pending status bit of the Interrupt Identity Register (IIR) is set. Note that this setting of an interrupt identity bit on the falling edge of the status bit is contrary to the general definition of interrupt status bits.

5.4.5 Display Pipe A,B VBLANK

These bits are set on the leading edge of the selected Display Pipe’s VBLANK signal.

5.4.6 User Interrupt

This bit is set in response to the execution of an MI_USER_INTERRUPT command. The Command Parser will continue parsing after processing that command. If a user interrupt is currently outstanding (set in the ISR) this packet has no effect.

Programming Note: User interrupts can be used to notify software of the progress of instruction parsing past the MI_USER_INTERRUPT instruction. Standard rules regarding instruction parsing versus instruction retirement documented in the section of this chapter still apply. In particular, user interrupts can be inserted into the command stream but effectively disabled for "normal operation" via the IMR and HWSTAM registers. Whenever software requires the notification afforded by the user interrupts, it can unmask this bit.

5.4.7 PIPE_CONTROL Notify Interrupt

This bit is set when a PIPE_CONTROL command with the **Notify Enable** bit set reaches the end of the pipeline and all required cache flushes have occurred.

5.4.8 Display Port Interrupt

This bit is set on a hot plug event. See the *Display Registers* chapter for details.

5.5 Hardware Status Writes

The graphics device supports the writing of the hardware status (ISR) bits into memory for optimized access from software. Software can select which (if any) status bits will trigger the write of the ISR contents to memory using the Hardware Status Mask (HWSTAM) register. Writing a '0' to a defined bit position in the HWSTAM register will cause any change (0 → 1 or 1→ 0) in the corresponding ISR bit to trigger the write. The complete ISR contents will be written to DWord offset 0 of the hardware status page, located at the address programmed via the Hardware Status Page Address Register (HWS_PGA).

5.6 Interrupts

The graphics device supports the generation of an interrupt. This interrupt can be raised in response to one or more internal interrupt status conditions. Which interrupt status conditions are allowed to raise an interrupt is programmed via the Interrupt Mask Register (IMR) and Interrupt Enable Register (IER). The IMR is used to selectively "unmask" hardware status bits as to allow them to be reported in the Interrupt Identity Register (IIR). The IER holds a set of interrupt enable bits corresponding to each bit of the IIR – setting bits in the IER will allow interrupts to be generated by the corresponding bits in the IIR.

5.7 Errors

The graphics device supports the hardware detection of a number of *operational* errors. Operational errors occur out of the immediate control of driver software and must be anticipated and tolerated to the extent required by the relevant APIs. Software must therefore support the detection and proper handling of all relevant operational errors. [Reserved content]

5.7.1 Error Reporting

Regardless of the error classification, all errors funnel through the **Master Error** bit of the Interrupt Control Registers. This bit can be used to raise a device interrupt or trigger a hardware status write operation. (Needless to say it can also be polled directly, though this is clearly discouraged). Refer to Interrupt Control Registers in the *Memory Interface Registers* chapter for more information.

There are three registers dedicated to control, detect, and clear hardware error status conditions in a similar fashion to the Interrupt Control Registers. All three error registers share a common error status bit definition.

The Error Status Register (ESR) holds the actual error status bits (each of which may be the logical OR of "source" error bits in various functional registers). The Error Mask Register (EMR) is used to select which error status bit(s) are reported in the Error Identity Register (EIR). The EIR holds the "persistent" values of the unmasked error status bits, and is also used to clear error status conditions. Any bits set in the EIR will raise the Master Error interrupt status condition.

5.7.2 Page Table Errors

The following tables describe the various sources and types of Page Table Errors.

Table 5-3. Page Table Error Types

Error	Description	Streams
Invalid GTT PTE	In the process of mapping an address, the MI encountered a GTT PTE that was marked "Invalid". This would be the result of a programming error.	All (See Table 4-1)
Invalid TLB Miss	An unexpected TLB miss (detected at GTT request time) was encountered (e.g., during Display/Overlay/Sprite access).	Display, Overlay
Invalid PTE Data	Mapping to the physical page specified in the PTE is not permitted (e.g., a page in PAM, SMM or over the top of memory, etc.). This is the result of a programming error.	Host
Invalid Tiling	A tiling parameter was found inconsistent with the current operation. This includes the use of Y-Major tiling in the Render/Display/Overlay streams. This is the result of a programming error. This is detected during GTT request.	Blt, Display, Overlay

Note that Page Table Errors cannot be cleared. A device reset is required.

5.7.2.1 Page Fault Handling

A page fault happens when a GTT entry is read to be used, and it's valid bit is 0 (Bit 0 of the GTT DW)

When GAM encounters a page fault, it will record the faulty address into a register, and invalidate the cycle(s) that use that page translation. Invalidating a cycle means asserting the invalid bit in the GAM-CI interface. This bit commands the Super Queue to cancel an invalid write, and return all zeros for an invalid read.

Since a page fault is not recoverable for the client that generates it, the invalid cycles are lost (in the sense that intended writes will end up not happening, and reads will return garbage = all zeros)

In order to simplify GAM design, an invalid GTT entry will be loaded into the TLB, and will continue behave as a valid one, with the exception of the invalid bit generation.

This means that it is possible to have a TLB hit, which has an invalid GTT saved for the physical address, which will always genrate invalid cycles.

Register Recording:

For each engine (Graphics, Media, and Blitter), there is a register that will be used for recording the first page fault that occurs for that engine. Subsequent faults will be treated as described above, but will not be recorded, until the valid bit of that register is cleared by Software (Valid bit is set when the first fault for that engine happens). This will provide a lead to Software on when things started going wrong.

Graphics Engine Fault Register Definition:

Address Offset: 4094-4097h

Default Value: XXXXXXXxxx0h

Access: RO; RW;

Size: 32 bits

GFX Arbiter Page Fault Register

Bit	Access	Default Value	RST/PWR	Description
31:12	RW	x	Core	**Virtual Address of Page Fault:** This is the original Address of the Page that generated the First Page fault for this engine. This value is locked and not updated on subsequent faults, until the valid bit of this register is cleared by SW
11	RW	x	Core	**GTT Sel:** This bit indicates if the valid bit happened while using PPGTT or GGTT: 0 – PPGTT, 1 - GGTT This value is locked and not updated on subsequent faults, until the valid bit of this register is cleared by SW
10:3	RW	x	Core	**SRCID of Page Fault:** This is the Source ID of the unit that requested the cycle that generated the First Page fault for this engine. This value is locked and not updated on subsequent faults, until the valid bit of this register is cleared by SW
2:1	RO	x	Core	**RESERVED.**
0	RW	0	Core	**Valid Bit:** This bit indicates that the first fault for this engine has been recorded. It can only be cleared by SW, which will also clear the other fields.

Media Engine Fault Register Definition:

Address Offset: 4194-4197h

Default Value: XXXXXXXxxx0h

Access: RO; RW;

Size: 32 bits

MEDIA Arbiter Page Fault Register

Bit	Access	Default Value	RST/PWR	Description
31:12	RW	x	Core	**Virtual Address of Page Fault:** This is the original Address of the Page that generated the First Page fault for this engine. This value is locked and not updated on subsequent faults, until the valid bit of this register is cleared by SW
11	RW	x	Core	**GTT Sel:** This is the original Address of the Page that generated the First Page fault for this engine. This value is locked and not updated on subsequent faults, until the valid bit of this register is cleared by SW
10:3	RW	x	Core	**SRCID of Page Fault:** This is the Source ID of the unit that requested the cycle that generated the First Page fault for this engine. This value is locked and not updated on subsequent faults, until the valid bit of this register is cleared by SW
2:1	RW	x	Core	**RESERVED.**
0	RW	0	Core	**Valid Bit:** This bit indicates that the first fault for this engine has been recorded. It can only be cleared by SW, which will also clear the other fields.

Blitter Engine Fault Register Definition:

Address Offset: 4294-4297h

Default Value: XXXXXXXxxx0h

Access: RO; RW;

Size: 32 bits

BLT Arbiter Page Fault Register

Bit	Access	Default Value	RST/P WR	Description
31:12	RW	x	Core	**Virtual Address of Page Fault:** This is the original Address of the Page that generated the First Page fault for this engine. This value is locked and not updated on subsequent faults, until the valid bit of this register is cleared by SW
11	RW	x	Core	**GTT Sel:** This is the original Address of the Page that generated the First Page fault for this engine. This value is locked and not updated on subsequent faults, until the valid bit of this register is cleared by SW
10:3	RW	x	Core	**SRCID of Page Fault:** This is the Source ID of the unit that requested the cycle that generated the First Page fault for this engine. This value is locked and not updated on subsequent faults, until the valid bit of this register is cleared by SW
2:1	RW	x	Core	**RESERVED.**
0	RW	0	Core	**Valid Bit:** This bit indicates that the first fault for this engine has been recorded. It can only be cleared by SW, which will also clear the other fields.

5.7.3 Clearing Errors

For operational errors, software is responsible for taking the proper steps to recover from the error and then clearing the error indication. The actions required to recover from operational errors may be discussed in the various functional areas (not here). See the Hardware-detected Error Bit Definitions in *Memory Interface Registers* for more details. This subsection describes the actions required to clear the error indication.

In order to clear operational errors, software is responsible for clearing the error condition from the source, working back to the Master Error bit. Typically this will entail the following sequence.

- First the primary source of the error must be cleared. This requires clearing the functional register(s) containing the source error indication.

- Next, clear the particular error status bit by writing a '1' to the appropriate bit of the Error Identity Register (EIR). This will clear the error status bit in the Error Status Register (ESR). If multiple errors are present, all error status bits should be cleared simultaneously.

- Next, clear the Master Error interrupt status bit by writing a '1' to the Master Error bit of the Interrupt Identity Register (IIR).

Note: Page Table Errors cannot be cleared.

5.8 Rendering Context Management

The graphics device operation (rendering, etc.) is controlled via the settings of numerous hardware state variables. These state variables are divided into *global state* and *context state*.

There is only one copy of global state variables, and changing the settings of these variables requires explicit programming of the state variables. Examples of global state include:

- MI registers (HWSTAM, Ring Buffer, etc.) <u>with the exception of those listed in the next paragraph</u> (i.e, registers listed there *are* saved/restored)

- Configuration registers

- Display programming registers

On the other hand, context state is associated with a specific *context*, where switching to that context causes that context's state to be restored. While the associated context is active, the state variables and registers can be programmed via the command stream. Examples of context state include the PIPELINE_STATE_POINTERS command and most non-pipelined state. The following MI registers are considered part of context state and thus saved/restored with context:

- INSTPM

- CACHE_MODE_0

- CACHE_MODE_1

- MI_ARB_STATE

- 3D Pipeline Statistics Registers

The graphics device supports both a *hardware context* and *logical contexts*. The multiple logical context support provides robust rendering context support by swapping contexts to/from memory.

5.8.1 Multiple Logical Rendering Contexts

The graphics device supports multiple *logical rendering contexts* stored in Main Memory. Logical rendering contexts are referenced via a 2KB-aligned *Logical Context Address*.

The maximum size of a logical context entry (which is information required by the driver to allocate contexts) is currently 2K bytes. For forward compatibility, the maximum size of a logical context entry should be supplied to the drivers via a VBIOS mechanism as opposed to being hardcoded in the driver.

The actual size of a logical rendering context is the amount of data stored/restored during a context switch and is measured in 64B cache lines.

The format of the logical rendering context in memory is considered device-dependent; software must not attempt to modify the contents of a logical rendering context directly. This restriction is motivated by forward compatibility concerns because the location and definition of fields may change between implementations.

5.8.1.1 Current Context IDs

The ring buffer has an associated *Current Context ID* (CCID) register. The CCID includes a Logical Pipeline Context Address (LPCA).

The CCID for a ring buffer is set during the processing of the new MI_SET_CONTEXT command from that ring. The MI_SET_CONTEXT command provides a new CCID value (LPCA) to be loaded into the CCID register for the associated ring buffer. The MI_SET_CONTEXT command also contains a Restore Inhibit bit used to optionally inhibit the restoration (loading) of the new rendering context. This bit must be used during context initialization to avoid the loading of uninitialized (garbage) context data from memory. Failure to do so leads to UNDEFINED operation.

The initial values of the CCIDs are UNDEFINED. The first time a valid CCID is set from a ring buffer, the normal context save operation will be suppressed, as the previous CCID is invalid.

5.8.1.2 Intra-Ring Context Switch

Within a specific ring buffer, a new logical rendering context is specified via the MI_SET_CONTEXT command. Note that MI_SET_CONTEXT commands are permitted only within a ring buffer (not within a batch buffer).

As part of the execution of the MI_SET_CONTEXT command from within a ring buffer, the Logical Pipeline Context Address fields of the CCID register and MI_SET_CONTEXT command are compared. If they differ (or the CCID register is uninitialized), a rendering context switch operation will be performed, which includes:

1. If the CCID contents are valid, a context save operation will be performed. The contents of the HW context will be saved in memory starting at the Logical Pipeline Context Address specified in the CCID.

2. If the Restore Inhibit command field is not set, a context restore operation will be performed. Here the logical context values are read starting from the Logical Pipeline Context Address field of the command and used to set the internal HW context.

3. The relevant contents of the command will be loaded into the appropriate CCID register. (This occurs irrespective of the LPCA comparison result). At this point, the ring buffer has switched to using the new logical rendering context.

5.8.1.3 Logical Rendering Context Creation and Initialization

5.8.1.3.1 Rendering Context Creation Rules

1. Software only knows the **size** of the logical rendering context (2KB), for allocation purposes.

2. Given (1), software does **not** know the format of the context, and therefore is not allowed to write any portion of a logical rendering context. Software can, however, copy/move entire logical context blocks.

3. Given (2), software must never restore (load) a logical rendering context from memory that has not been previously stored by HW. I.e., software must never attempt to initialize a context itself and then cause it to be loaded. Breaking this rule causes UNDEFINED operation (as in the hang seen in BDG validation).

4. Initialization software must write **all** HW context variables with legal values before the first rendering context can be saved (this must be done before you can perform any rendering anyways). Given this, and the obvious rule that software must never program illegal state values, guarantees that the HW context will forever remain valid (and therefore be available to store into a logical rendering context). Note that software-visible context variables include 3D state, Blt register state, etc.

5.8.1.3.2 Context Initialization

Logical Rendering Contexts can be initialized (in memory) by software in the following way:

1. Issue an MI_SET_CONTEXT command w/ the **Restore Inhibit** bit set and the about-to-be-initialized logical pipeline context address. This will save the current rendering context and then change the LPCA to the new context (without loading it).

2. Use state commands to modify the context as desired.

3. Issue another MI_SET_CONTEXT command specifying some other LPCA (e.g., the previous one). This will cause the new context to be stored (initialized) in memory

5.8.1.4 Context Save

A context save will occur anytime all of the following apply:

- A rendering context switch occurs as a result of the execution of MI_SET_CONTEXT

- the CCID of the current context (CCID register of current ring) and the new CCID (the CCID register of the newly selected ring or the new CCID in the MI_SET_CONTEXT command) differ OR an MI_SET_CONTEXT with the “Force Restore” bit set initiated the context switch

- the current CCID is valid (has been previously set)

The current rendering context will be written out to memory starting at the LPCA in the format described by Logical Context Layout in *Memory Data Formats*. Note that this includes a limited number of Memory Interface Registers whose values are saved by embedding them in an MI_LOAD_REGISTER_IMM command that is written out to memory.

The Optional Extended Context will also be written if the Extended Save Enable bit is set in the current CCID register. Context saves DO NOT modify pipelined state stored in memory.

5.9 Reset State

This section describes the state of the programming interface following a hardware reset. Refer to the individual register definitions for details on reset (default) settings.

- The settings of the hardware context state variables are UNDEFINED. Software must program all state variables prior to their use in rendering.
- The ring buffer is disabled.
- All interrupts and error status bits are “masked” (disabled). All interrupts are disabled via IER. There will be no HW activity to cause any hardware/interrupt status bits to be set.
- The Hardware Status Page is located at 1FFFF000h (though HW status writes are effectively disabled)
- All FENCE registers are INVALID
- The GTT is disabled (accesses other than CPU reads, cursor and VGA reads will generate an error).
- All INSTDONE bits are set (“DONE”).
- The NOPID register is 0.
- All command groupings are enabled (via INSTPM)

Revision History

Revision Number	Description	Revision Date
1.0	First 2011 OpenSource edition	May 2011

§§

Intel® OpenSource HD Graphics Programmer's Reference Manual (PRM)

Volume 1 Part 3: Graphics Core – Memory Interface and Commands for the Render Engine (SandyBridge)

For the 2011 Intel Core Processor Family

May 2011

Revision 1.0

NOTICE:

This document contains information on products in the design phase of development, and Intel reserves the right to add or remove product features at any time, with or without changes to this open source documentation.

Doc Ref #: IHD-OS-V1 Pt3 – 05 11

Creative Commons License

You are free to Share — to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

No Derivative Works. You may not alter, transform, or build upon this work.

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

Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The SandyBridge chipset family, Havendale/Auburndale chipset family, Intel® 965 Express Chipset Family, Intel® G35 Express Chipset, and Intel® 965GMx Chipset Mobile Family Graphics Controller may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order. I2C is a two-wire communications bus/protocol developed by Philips. SMBus is a subset of the I2C bus/protocol and was developed by Intel®.

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

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

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

Copyright © 2011, Intel Corporation. All rights reserved.

Contents

1. Render Engine Command Streamer 4

1.1 Registers in Render Engine 4
1.1.1 Introduction 4
1.1.2 Virtual Memory Control 6
1.1.3 Probe List Registers 9
1.1.4 Context Save Registers 11
1.1.5 Mode and Misc Ctrl Registers 13
1.1.6 RINGBUF — Ring Buffer Registers 39
1.1.7 Watchdog Timer Registers 46
1.1.8 Interrupt Control Registers 48
1.1.9 Logical Context Support 56
1.1.10 Pipelines Statistics Counter Registers 68
1.1.11 Performance Statistics Registers 75
1.2 Memory Interface Commands for Rendering Engine 95
1.2.1 Introduction 95
1.2.2 Software Synchronization Commands 95
1.2.3 MI_ARB_CHECK 96
1.2.4 MI_ARB_ON_OFF 97
1.2.5 MI_BATCH_BUFFER_END 98
1.2.6 MI_CONDITIONAL_BATCH_BUFFER_END 98
1.2.7 MI_BATCH_BUFFER_START 100
1.2.8 MI_CLFLUSH 103
1.2.9 MI_DISPLAY_FLIP 105
1.2.10 MI_FLUSH 109
1.2.11 MI_LOAD_REGISTER_IMM 111
1.2.12 MI_NOOP 113
1.2.13 Surface Probing 114
1.2.14 MI_REPORT_HEAD 114
1.2.15 MI_SEMAPHORE_MBOX 115
1.2.16 MI_SET_CONTEXT 117
1.2.17 MI_STORE_DATA_IMM 120
1.2.18 MI_STORE_DATA_INDEX 121
1.2.19 MI_STORE_REGISTER_MEM 123
1.2.20 MI_SUSPEND_FLUSH 125
1.2.21 MI_UPDATE_GTT 126
1.2.22 MI_USER_INTERRUPT 127
1.2.23 MI_WAIT_FOR_EVENT 128

1. Render Engine Command Streamer

[DevSNB-D2] On hard boot, the command streamer must be programmed as follows to work-around a known issue that affects power management. This is expected to be done in BIOS

1) Disable CSunit level clock gating
2) Reset Render pipe

1.1 Registers in Render Engine

1.1.1 Introduction

This chapter describes the memory-mapped registers associated with the Memory Interface, including brief descriptions of their use. The functions performed by some of these registers are discussed in more detail in the Memory Interface Functions, Memory Interface Instructions, and Programming Environment chapters.

The registers detailed in this chapter are used across the GEN6 family of products and are extentions to previous projects. However, slight changes may be present in some registers (i.e., for features added or removed), or some registers may be removed entirely. These changes are clearly marked within this chapter.

1.1.1.1 ARB_MODE – Arbiter Mode Control Register [DevSNB]

ARB_MODE – Arbiter Mode Control Register	
Register Type:	MMIO_CS
Address Offset:	4030h
Project:	DevSNB+
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Trusted Type:	1

Bit	Description
31:16	**Mask bits** Project: DevSNB+ Format: U16 Mask bits act as write enables for the bits in the lower bits of this register
15:9	**Reserved** Project: All Format: MBZ
8	**Reserved**
7:6	**Reserved** Project: All Format:

ARB_MODE – Arbiter Mode Control Register	
5:4	**Address Swizzling for Tiled-Surfaces** Project: All Format: U1 This register location is updated via GFX Driver prior to enabling DRAM accesses. Driver needs to obtain the need for memory address swizzling via DRAM configuration registers and set the following bits (in Display Engine and Render/Media access streams)

Value	Name	Description	Project
00	No address Swizzling	No address Swizzling	DevSNB+
01	Address bit[6] needs to be swizzled for tiled surfaces	Address bit[6] needs to be swizzled for tiled surfaces	DevSNB+
10		Reserved	DevSNB+
11		Reserved	DevSNB+

3:0	**Reserved**

1.1.1.2 ARB_WR_GAC_GAM3 – GAC_GAM WR Arbitration Register 3

ARB_WR_GAC_GAM3	
Register Type:	MMIO_CS
Address Offset:	43FCh
Pwrject:	DevSNB+
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Trusted Type:	1

Bit	Description
31:28	**Reserved**
27	Priority for entry 7
26:24	Goto field for entry 7 when request vector is 11.
23:21	Goto field for entry 7 when request vector is 10.
20:18	Goto field for entry 7 when request vector is 01.
17:15	Goto field for entry 7 when request vector is 00.
14:13	**Reserved**
12	Priority for entry 6
11:9	Goto field for entry 6 when request vector is 11.
8:6	Goto field for entry 6 when request vector is 10.
5:3	Goto field for entry 6 when request vector is 01.
2:0	Goto field for entry 6 when request vector is 00.

1.1.2 Virtual Memory Control

1.1.2.1 HWS_PGA — Hardware Status Page Address Register

HWS_PGA — Hardware Status Page Address Register	
Register Type:	MMIO_CS
Address Offset:	4080h
Project:	All
Default Value:	UUUU0000h
Access:	R/W
Size (in bits):	32
Trusted Type:	1

This register is used to program the 4 KB-aligned System Memory address of the Hardware Status Page used to report hardware status into (typically cacheable) System Memory. **[DevSNB]** This address in this register is translated using the Global GTT in memory. The mapping type of the GTT entry determines the snoop nature of the transaction to memory.

Bit	Description
31:12	**Address** Project: All Security: None Address: GraphicsAddress[31:12] This field is used by SW to specify Bits 31:12 of the 4 KB-aligned System Memory address of the 4 KB page known as the "Hardware Status Page". The Global GTT is used to map this page from the graphics virtual address to physical address **Programming Notes** If the **Per-Process Virtual Address Space** is set, HW requires that the status page is programmed to allow for the context switch status to be reported
11:0	**Reserved** Project: All Format: MBZ

The following table defines the layout of the Hardware Status Page:

DWord Offset	Description
0	**Interrupt Status Register Storage:** The content of the ISR register is written to this location whenever an "unmasked" bit of the ISR (as determined by the HWSTAM register) changes state.
3:1	**Reserved.** Must not be used.
4	**Ring Head Pointer Storage:** The contents of the Ring Buffer Head Pointer register (register DWord 1) are written to this location either as result of an MI_REPORT_HEAD instruction or as the result of an "automatic report" (see RINGBUF registers).
Fh:5h	**Reserved.** Must not be used.
10h-1Bh	**Context Status DWords.**
1Ch-1Eh	**Reserved.** Must not be used.
1Fh	**Last Written Status Offset.**
20h-3FFh	These locations can be used for general purpose via the MI_STORE_DATA_INDEX or MI_STORE_DATA_IMM instructions.

1.1.2.2 PP_DIR_BASE – Page Directory Base Register

PP_DIR_BASE – Page Directory Base Register	
Register Type:	MMIO_CS
Address Offset:	{DevSNB] Write offset: 0x2228 Read offset: 0x2518
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32
This register contains the offset into the GGTT where the (current context's) PPGTT page directory begins. This register is restored with context. The Page Directory Base Address is set by SW only by modifying the value of this register in the context image such that the new value is restored the next time the context runs. A write via MMIO to this register triggers the render pipe to fetch all PDs. **Programming Note:** The **MBC Driver Boot Enable** bit in **MBCTL** register must be set *before* this register is written to upon boot up (including S3 exit)	
Bit	**Description**

PP_DIR_BASE – Page Directory Base Register	
30:16	**Page Directory Base Offset** Project: All Default Value: 0h Format: U15 Range [0,GGTT Size in cachelines - 1] Contains the cacheline (64-byte) offset into the GGTT where the page directory begins.
15:1	**Reserved** Project: All Format: MBZ
0	**PD Load Busy** Project: DevSNB+ Format: Valid This is a read-only field that indicates if the page directories are currently being fetched and loaded.

1.1.2.3 PP_DCLV – PPGTT Directory Cacheline Valid Register

PP_DCLV – PPGTT Directory Cacheline Valid Register	
Register Type:	MMIO_CS
Address Offset:	2220h
Project:	All
Default Value:	0h
Access:	{DevSNB] Write only. Cannot read via MMIO
Size (in bits):	64

This register controls update of the on-chip PPGTT Directory Cache during a context restore. Bits that are set will trigger the load of the corresponding 16 directory entry group. This register is restored with context (prior to restoring the on-chip directory cache itself). This register is also restored when switching to a context whose LRCA matches the current CCID if the **Force PD Restore** bit is set in the context descriptor.

The context image of this register must be updated and maintained by SW; SW should not normally need to read this register.

This register can also effectively be used to limit the size of a processes' virtual address space. Any access by a process that requires a PD entry in a set that is not enabled in this register will cause a fatal error, and no fetch of the PD entry will be attempted

Bit	Description
63:32	**Reserved** Project: All Format: MBZ
31:0	**PPGTT Directory Cache Restore [1..32] 16 entries** Project: All Format: Array:Enable If set, the [1st..32nd] 16 entries of the directory cache are considered valid and will be brought in on context restore. If clear, these entries are considered invalid and fetch of these entries will not be attempted.

1.1.3 Probe List Registers

Surface probing is a procedure performed at the beginning of a rendering sequence (command buffer) to verify that all required surfaces in a process' virtual address space are actually present in physical memory prior to beginning the sequence. A different process can then be switched to and run while the required surfaces are being brought into memory (by SW). The register work in concert with the probe commands (see Memory Interface Commands for Rendering) to provide this interface. "Slots" are the designated places in a processes' context image where probes (surface base addresses) are stored. The stored probes are used by SW to determine which surfaces a context requires, and are also used by HW to re-validate that surfaces are resident upon a context restore.

See MI_PROBE in Memory Interface Commands for Rendering for more details.

Note these register should only be used when Surface Fault Enable bit is set in GFX_MODE.

1.1.3.1 PRBL_SF – Probe List Slot Fault Register

PRBL_SF – Probe List Slot Fault Register	
Register Type:	MMIO_CS
Address Offset:	2680h {DevSNB]
Project:	All
Default Value:	0000 0000h
Access:	RO
Size (in bits):	64

This register contains the fault bits for the probe slots, one bit for each cacheline of the 1024 probe slot memory area. It cannot be directly written by SW. The image of this register in the per-process HW status page can be read after a context switch (due to surface fault) to determine which cachelines of the probe list contain faulting probes. This register is saved with context. It is not restored but recomputed while re-validating the probe list on a context restore.

Bit	Description
63:0	**Slot Fault Line 63:0** Project: All Format: Array:Enable If set, indicates that the corresponding probe list cacheline (in memory) contains a probe that has faulted.

This interface is used to signal page faults that occur during access of per-process virtual graphics memory. A fault of this nature will stall the 3D/Media pipeline behind the fault, and all new TLB requests from anywhere in the pipeline will be stalled. Faults are recorded in a fault log consisting of 32 fault slots. Page faults are non-recoverable events and will cause hardware to hang.

1.1.3.2 PP_PFIR – PPGTT Page Fault Indication Register

PP_PFIR – PPGTT Page Fault Indication Register	
Register Type:	MMIO_CS
Address Offset:	4510h
Project:	All
Default Value:	0000 0000h
Access:	R/WC
Size (in bits):	32
This register contains the flags for page faults. All bits should be cleared at once by writing FFFFFFFFh to this register once all faults have been serviced. No additional bits of this register will become set (signaling additional faults) between the time the page fault interrupt has been sent to the host and the time the host clears the **Fault In Service** bit indicating it is done servicing faults	
Bit	**Description**
31:0	**Page Fault [31:0]** Project: All Format: Array:Flag Fault indicator for page fault log index [31:0]. When set, this flag indicates that a page fault is outstanding. The invalid page address that was accessed can be read from fault entry [31:0]. SW should clear this bit by writing a '1' to it to indicate to HW that the fault has been serviced (the page has been mapped and should now be valid).

1.1.3.3 PP_PFD[0:31] – PPGTT Page Fault Data Registers

PP_PFD[0:31] – PPGTT Page Fault Data Registers	
Register Type:	MMIO_CS
Address Offset:	4580h
Project:	All
Security:	None
Default Value:	0000 6820h
Access:	RO
Size (in bits):	32
The GTT Page Fault Log entries can be read from these registers. 4580h-4583h: Fault Entry 0 … 45FCh-45FFh: Fault Entry 31	
Bit	**Description**
31:12	**Fault Entry Page Address** Project: All Address: GraphicsAddress[31:12] This RO field contains the faulting page address for this Fault Log entry. This field will contain a valid fault address only if the bit in the GTT Page Fault Indication Register corresponding with the address offset of this entry is set.
11:0	**Reserved** Project: All Format: MBZ

1.1.4 Context Save Registers

1.1.4.1 BB_PREEMPT_ADDR—Batch Buffer Head Pointer Preemption Register

BB_PREEMPT_ADDR—Batch Buffer Head Pointer Preemption Register	
Register Type:	MMIO_CS
Address Offset:	2148h
Project:	All
Default Value:	0000 0000h
Access:	RO
Size (in bits):	32

This register contains the current DWord-aligned Graphics Memory Address MI_ARB_CHECK in a batch buffer where the UHPTR register was valid. The value of the pointer below will be the address of the MI_ARB_CHECK that caused the head pointer to move.

This register is invalid if the previous preemption due to an MI_ARB_CHECK executed in the ring.

Programming Restriction:
This register should NEVER be programmed by driver, this is for HW internal use only.

Bit	Description
31:2	**Batch Buffer Head Pointer** Project: All Format: GraphicsAddress[31:2] This field specifies the DWord-aligned Graphics Memory Address MI_ARB_CHECK in a batch buffer where the UHPTR register was valid.
1:0	**Reserved** Project: All Format: MBZ

1.1.4.2 BB_START_ADDR—Batch Buffer Start Head Pointer Register

BB_START_ADDR—Batch Buffer Start Head Pointer Register	
Register Type:	MMIO_CS
Address Offset:	2150h
Project:	All
Default Value:	0000 0000 0000 0000h
Access:	RO
Size (in bits):	32

This register contains the address specified in the last MI_START_BATCH_BUFFER command.

Programming Restriction:
This register should NEVER be programmed by driver, this is for HW internal use only.

Bit	Description
31:2	**Batch Buffer Start Head Pointer** Project: All Format: GraphicsAddress[31:2] This field specifies the DWord-aligned Graphics Memory Address where the last initiated Batch Buffer starting address.
1:0	**Reserved** Project: All Format: MBZ

1.1.4.3 BB_OFFSET—Batch Buffer Address Difference Register

BB_ADDR_DIFF—Batch Address Difference Register	
Register Type:	MMIO_CS
Address Offset:	2154h
Project:	All
Default Value:	0000 0000 0000 0000h
Access:	RO
Size (in bits):	32

This register contains the difference between the start of the last batch and where the last initiated Batch Buffer is currently fetching commands.

Programming Restriction:
This register should NEVER be programmed by driver, this is for HW internal use only.

Bit	Description
31:2	**Batch Buffer Address Difference** Project: All Format: GraphicsAddress[31:2] This field specifies the DWord-aligned difference between the starting address of the batch buffer and where the last initiated Batch Buffer is currently fetching commands.
1:0	**Reserved** Project: All Format: MBZ

1.1.5 Mode and Misc Ctrl Registers

1.1.5.1 MI_MODE — Mode Register for Software Interface

MI_MODE — Mode Register for Software Interface

Register Type: MMIO_CS
Address Offset: 209Ch
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 32

The MI_MODE register contains information that controls software interface aspects of the Memory Interface function.

Bit	Description
31:16	**Masks** Format: Mask[15:0] A "1" in a bit in this field allows the modification of the corresponding bit in Bits 15:0
15	**Suspend Flush** Project: DevSNB Default Value: 0h DefaultVaueDesc Format: U1 FormatDesc

Value	Name	Description	Project
0h	No Delay	HW will not delay flush, this bit will get cleared by MI_SUSPEND_FLUSH as well	All
1h	Delay Flush	HW will delay the flush because of sync flush or VTD regimes until reset, this bit will get set by MI_SUSPEND_FLUSH as well	All

Programming Notes	Project
This should only be written to from the ring using MI_SUSPEND_FLUSH. It is considered undefined if written by software through MMIO	All

14

Async Flip Performance mode

Project: All

Default Value: 0h

Format: U1

[DevSNB] This bit must be set to '1'

Value	Name	Description	Project
0h	Performance mode enabled	The stall of the flip event is in the windower	All
1h	Performance mode disabled	The stall of the flip event is in the command stream	All

13

Flush Performance mode

Project: All

Default Value: 0h

Format: U1

Value	Name	Description	Project
0h	run fast restore	No NonPipelined SV flush.	All
1h	run slow legacy restore	With NonPipelined SV flush.	All

12

MI_FLUSH Enable

Project: DevSNB

Default Value: 0h DefaultVaueDesc

Format: Enable

PIPE_CONTROL is a superset of MI_FLUSH. Since MI_FLUSH is redundant, it will be removed in future projects beyond GT. By default, it is disabled

Value	Name	Description	Project
0h	Disable	If an MI_FLUSH is parsed with this bit disabled, the parser will stall and the parser error bit will be set in the ESR creating an interrupt	DevSNB
1h	Enable	If an MI_FLUSH is parsed with this bit enabled, the parser will execute the legacy command according to the bspec	DevSNB

11

Invalidate UHPTR enable Project: All Format: Enable

If bit set H/W clears the valid bit of UHPTR (2134h, bit 0) when current active head pointer is equal to UHPTR.

MI_MODE — Mode Register for Software Interface

10	**Reserved** Project: All Format: MBZ
9	**Rings Idle** Project: All Default Value: 0h Format: U1 Read Only Status bit

Value	Name	Description	Project
0h	Not Idle	Parser not Idle or Ring Arbiter not Idle.	All
1h	Idle	Parser Idle and Ring Arbiter Idle.	All

Programming Notes	Project
Writes to this bit are not allowed.	All

8 **Stop Rings**

Project: All

Default Value: 0h

Format: U1

Value	Name	Description	Project
0h		Normal Operation.	All
1h		Parser is turned off and Ring arbitration is turned off.	All

Programming Notes	Project
Software must set this bit to force the Rings and Command Parser to Idle. Software must read a “1” in Ring Idle bit after setting this bit to ensure that the hardware is idle.	All
Software must clear this bit for Rings to resume normal operation.	All

7 **Vertex Shader Cache Mode**

Project: All

Default Value: 0h

Format: U1

Value	Name	Description	Project
0h	Non-LRA	Non-LRA mode of allocation. Vertex shader cache is allocated on the basis of the reference count of individual vertices	All
1h	LRA	LRA mode of allocation. Used for validation purposes.	All

MI_MODE — Mode Register for Software Interface

6 — **Vertex Shader Timer Dispatch Enable**

Project: All

Default Value: 0h

Format: Enable

Value	Name	Description	Project
0h	Disable	Disable the timer for dispatch of single vertices from the vertex shader. Vertex shader will try to collect 2 vertices before a dispatch	All
1h	Enable	Enable the timer for dispatch of single vertices. Dispatch a single vertex shader thread after the timer expires.	All

Programming Notes	Project
To avoid deadlock conditions in hardware this bit needs to be set for normal operation.	All

5 — **Reserved** Project: All Format: MBZ

4 — **Enable Software Element Configuration**

Project: DevSNB+

Default Value: 0h

Format: Enable

Value	Name	Description	Project
0h	Disable	Hardware will choose how to pass elements down the quad pipe of the Vertex Fetch	DevSNB
1h	Enable	Software will be able to choose which configuration to pass elements down the Vertex Fetch pipeline. See the 3D_VERTEX_ELEMENTS command in the 3D_pipeline chapter for more details.	DevSNB

3:1 — **Reserved** Project: All Format: MBZ

Read/Write

0 — **Mask IIR disable** Project: All Format: Disable

Mask IIR disable. Nominally the Interrupt controller masks interrupts in the IIR register if an interrupt acknowledge from the 3gio interface is pending. Setting this bit to a “1” allows interrupts to be visible to the interrupt controller while an interrupt acknowledge is pending.

1.1.5.2 GFX_MODE – Graphics Mode Register

GFX_MODE

Register Type: MMIO
Address Offset: 2520h {DevSNB]
Project: All
Default Value: 00000800h {DevSNB]

Access: R/W
Size (in bits): 32
Trusted Type: 1

This register contains a control bit for the PPGTT functions. This register is not saved/restored with context. This register is not reset with single-engine GFX reset; it is only reset by a global graphics reset (all engines including display).

Bit	Description
31:16	**Mask Bits** Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
15	**Reserved**
14	**Reserved** Project: All Format: MBZ
13	**Flush TLB invalidation Mode** Project: All Format: U1 This field controls the invalidation if the TLB cache inside the hardware. When *enabled* this bit limits the invalidation of the TLB only to batch buffer boundaries or to pipe_control commands which have the TLB invalidation bit set. If *disabled*, the TLB caches are flushed for every full flush of the pipeline. [DevSNB A] This bit must be '0' **Value** / **Name** / **Description** / **Project** 0h / Disabled / when '0', the TLB caches are flushed for every full flush of the pipeline. / {DevSNB] 1h / Enabled / when '1' only send TLB inv on batch buffer boundaries or when PIPE_CONTROL w/ TLB inv bit is set / {DevSNB]
12	**Surface Fault Enable** Project: All Format: U1 When set, surface and page fault will be handled in HW. It is undefined to use MI_PROBE and MI_UNPROBE if this bit is clear 0: surface/page fault handling disabled (default)

GFX_MODE

11	**Replay Mode** Project: All Default Value: 1h midtriangle Mask: MMIO(0x2000)#16 Format: U1 Context Switch Granularity This field controls the granularity of the replay mechanism when coming back into a previously preempted context.

Value	Name	Description	Project
0h	mid-triangle preemption	Super span Level. Pipeline is not flushed. This implies commands parsed are executed speculatively and may not complete before a context switch.	All
1h	mid-cmdbuffer preemption	Drawcall Level. Pipeline is flushed before switching to the next context. Commands parsed are commited to completing before a context switch	All

Programming Notes

- A fixed function pipe flush is required before modifying this field

Unless pre-emption at a mid-triangle is required the bit must be set.

10	**Reserved**

GFX_MODE

9	**Per-Process GTT Enable** Project: All Default Value: 0h Disabled Format: Enabled Per-Process GTT Enable

Value	Name	Description	Project
0h	PPGTT Disable	When clear, the Global GTT will be used to translate memory access from designated commands and for commands that select the PPGTT as their translation space in single-context scheduling mode.	All
1h	PPGTT Enable	When set, the PPGTT will be used to translate memory access from designated commands and for commands that select the PPGTT as their translation space. The **PD Offset** and **PD Cacheline Valid** registers must be set in all pipes (blitter, MFX, render) before any workload is submitted to hardware. This mode enables support for big pages (32k)	All

Programming Notes:

- **[DevSNB A/B {W/A}]:** If RC6 is enabled and PPGTT mode is used, software must program the **CTX_WA_PTR (0x2058)** on boot for power context. Inside the work-around batch memory, there must be several MI_LOAD_REGISTER_IMM (LRI) commands to reload the PD Offset.
  - LRI address = 0x02228, data = Render PD base addr (statically defined)
  - LRI address = 0x12228, data = MFX PD base addr (statically defined)
  - LRI address = 0x22228, data = Blitter PD base addr (statically defined)
- [**DevSNB]** PPGTT memory writes by MI_* (such as MI_STORE_DATA_IMM) and PIPE_CONTROL are not supported.

8	**Reserved** Project:
7:0	**Reserved** Project: All Format: MBZ

1.1.5.3 GT_MODE – GT Mode Register [DevSNB+]

GT_MODE – GT Mode Register

Register Type: MMIO_CS[DevSNB]

Address Offset: 20D0h[DevSNB]

Project: DevSNB+
Default Value: 00000000h
Access: R/W
Size (in bits): 32
Trusted Type: 1

This Register is used to control the 6EU and 12EU configuration for SNB.
Write 0x01FF01FF to this register enables the 6EU mode.
[DevSNB A] Software must perform a read-modify-write sequence to update register after initial value is written. Also, every write must have value [31:16] = 0xFFFF

Bit	Description
31:16	**Mask Bits** Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
15	**Reserved**
14:11	**Reserved** :
14:13	**Reserved**
14:13	**Reserved** Project: Format: MBZ
12:11	**Reserved**
10	**Reserved**
9	**WIZ Hashing Mode High Bit** Project: DevSNB-B+ Default Value: 1h Format: U1 This field adds additional hashing modes in combination with the **WIZ Hashing Mode** field. The Value column in the table below refers to this field (high bit) and the **WIZ Hashing Mode** field (low bit). *This field is don't care if the Hashing Disable bit is set.*

Value	Name	Description	Project
0h		8x8 Checkerboard hashing	DevSNB-B+
1h		8x4 Checkerboard hashing	DevSNB-B+
2h		16x4 Checkerboard hashing	DevSNB-B+
3h		Reserved	

GT_MODE – GT Mode Register

8 — Full Rate Sampler Disable

Project: DevSNB
Default Value: 0h
Format: Enable

This field configures the sampler rate.

Value	Name	Description	Project
0h	Disable	Full rate sampler	DevSNB
1h	Enable	Half rate sampler	DevSNB

7 — WIZ Hashing Mode

Project: DevSNB+
Default Value: 0h
Format: U1

This field configures the Hashing mode in Windower. For [DevSNB-B+], the **WIZ Hashing Mode High Bit** field is combined with this field to enable additional modes.

This field is don't care if the Hashing Disable bit is set.

Value	Name	Description	Project
0h		16x4 Checkerboard hashing	DevSNB
1h		8x4 Checkerboard hashing	DevSNB

6 — Reserved

5 — TD Four Row Dispatch Disable

Project: DevSNB
Default Value: 0h
Format: Enable

This field configures the number of rows TD dispatchs thread into.

Value	Name	Description	Project
0h	Disable	TD dispatchs to all 4 rows	DevSNB
1h	Enable	TD dispatchs to only row0 and row1	DevSNB

4 — Full Size URB Disable

Project: DevSNB
Default Value: 0h
Format: Enable

This field configures the size of the URB.

Value	Name	Description	Project
0h	Enable	Full size URB	DevSNB
1h	Disable	Half size URB	DevSNB

GT_MODE – GT Mode Register

3 — **Full Size SF FIFO Disable**

Project: DevSNB
Default Value: 0h
Format: Enable

This field configures the size of the FIFO between SF and PSD.

Value	Name	Description	Project
0h	Enable	Full size SF FIFO	DevSNB
1h	Disable	Half size SF FIFO	DevSNB

2 — **Reserved** Project: All Format: MBZ

1 — **VS Quad Thread Dispatch Disable**

Project: DevSNB
Default Value: 0h
Format: Enable

This field configures the number of dispatch ports in VS unit.

Value	Name	Description	Project
0h	Enable	Quad thread dispatch enabled for VS	DevSNB
1h	Disable	Quad thread dispatch disabled for VS	DevSNB

0 — **Reserved**

1.1.5.4 Cache_Mode_0— Cache Mode Register 0

Cache_Mode_0— Cache Mode Register 0

Register Type:	MMIO_CS[DevSNB]
Address Offset:	2120h [DevSNB]
Project:	All
Default Value:	0000 6820h [DevSNB]
Access:	R/W
Size (in bits):	32

This register is used to control the operation of the Render and Sampler L2 Caches. All reserved bits are implemented as read/write.

Before changing the value of this register, GFX pipeline must be idle i.e. full flush is required

This Register is saved and restored as part of Context.

Bit	Description
31:16	**Masks** Format: Mask[15:0] A "1" in a bit in this field allows the modification of the corresponding bit in Bits 15:0.
15	**Sampler L2 Disable** Project: All Default Value: 0h Format: Disable

Value	Name	Description	Project
0h		Sampler L2 Cache Enabled.	All
1h		Sampler L2 Cache Disabled all accesses are treated as misses.	All

Errata	Description	Project
BWT012	Setting this bit is UNDEFINED.	DevBW-A,B

14:10	**Reserved** Project: All Format: MBZ

Cache_Mode_0— Cache Mode Register 0

9	**Sampler L2 TLB Prefetch Enable** Project: All Default Value: 0h Format: Enable

Value	Name	Description	Project
0h		TLB Prefetch Disabled	All
1h		TLB Prefetch Enabled	All

8	**Reserved**
7:6	**Sampler L2 Request Arbitration** Project: All Default Value: 0h Format: U2

Value	Name	Description	Project
00		Round Robin	All
01		Fetch are Highest Priority	All
10		Constants are Highest Priority	All
11		Reserved	All

5	**STC Eviction Policy** Project: All Format: Disable If this bit is set, STCunit will have LRA as replacement policy. The default value i.e. (when this bit is reset) indicates that non-LRA eviction policy. This bit must be reset. LRA replacement policy is not supported.
4	**RCC Eviction Policy** Project: [DevSNB+] Format: Disable If this bit is set, RCCunit will have LRA as replacement policy. The default value i.e. (when this bit is reset) indicates that non-LRA eviction policy. This bit must be reset. LRA replacement policy is not supported.
3	**Reserved**
2	**Reserved** Project: Format: MBZ
2	**Reserved**
1	**Disable clock gating in the pixel backend** Project: All Format: Disable MCL related clock gating is disabled in the pixel backend. Before setting this bit to 1, the instruction/state caches must be invalidated. [DevSNB:{WKA}]

Cache_Mode_0— Cache Mode Register 0

0	**Render Cache Operational Flush Enable** Project: [All] Default Value: 0h Format: Enable

Value	Name	Description	Project
0h	Disable	Operational Flush Disabled (recommended for performance when not rendering to the front buffer)	All
1h	Enable	Operational Flush Enabled (required when rendering to the front buffer)	All

Errata	Description	Project
	This bit must be 0. Operational Flushes are not supported in [DevSNB]. SW must flush the render target after front buffer rendering.	[DevSNB]

1.1.5.5 Cache_Mode_1— Cache Mode Register 1

Cache_Mode_1— Cache Mode Register 1

Register Type: MMIO_CS [DevSNB]

Address Offset: 2124h [DevSNB]

Project: All
Default Value: 0000 0180h
Access: Read/32 bit Write
Size (in bits): 32

Before changing the value of this register, GFX pipeline must be idle i.e. full flush is required.

This Register is saved and restored as part of Context.

Bit	Description
31:16	**Mask Bits for 15:0** Format: Mask[15:0] Must be set to modify corresponding data bit. Reads to this field returns zero.
15	**Reserved** Project: All Format: MBZ

Cache_Mode_1— Cache Mode Register 1

14	**Reserved**
13	**Reserved**
12	**HIZ Eviction Policy** Project: All Default Value: 0h Format: U1 If this bit is set, Hizunit will have LRA as replacement policy. The default value i.e. (when this bit is reset) indicates the non-LRA eviction policy. For performance reasons, this bit must be reset.

Value	Name	Description	Project
0h		Non-LRA eviction Policy	All
1		LRA eviction Policy	All

11	**DAP Instruction and State Cache Invalidate** Project: All Default Value: 0h Format: U1 When this field is set, DAP instruction and state caches (level 1 and level 2) are invalidated.

Value	Name	Description	Project
0h		Normal Cache operation.	All
1		Reserved	All

10	**Instruction Level 1 Cache and In-Flight Queue Disable** Project: All Default Value: 0h Format: Disable

Value	Name	Description	Project
0h		Cache is enabled.	All
1h		Reserved	All

Cache_Mode_1— Cache Mode Register 1

9

Instruction and State Level 2 Cache Fill Buffers Disable

Project: All

Default Value: 0h

Format: Disable

Value	Name	Description	Project
0h		Fill Buffers are enabled.	All
1h		Reserved	All

8:7

Sampler Cache Set XOR selection

Project: All

Default Value: 3h

Format: U2

These bits have an impact only when the Sampler cache is configured in 16 way set associative mode. If the cache is being used for immediate data or for blitter data these bits have no effect.

Value	Name	Description	Project
00	Default value	Default behavior to calculate set address, no XOR.	All
01	Scheme 1	New_set_mask[3:0] = Tiled_address[16:13] New_set[3:0] <= New_set_mask[3:0] ^ Old_set[3:0] Rationale: These bits can distinguish among 16 different equivalent classes of virtual pages. These bits also represent the lsb for tile rows ranging from a pitch of 1 tile to 16 tiles.	All
10	Scheme 2	New_set_mask[3] = Tiled_address[17] ^ Tiled_address[16] New_set_mask[2] = Tiled_address[16] ^ Tiled_address[15] New_set_mask[1] = Tiled_address[15] ^ Tiled_address[14 New_set_mask[0] = Tiled_address[14] ^ Tiled_address[13] New_set[3:0] <= New_set_mask[3:0] ^ Old_set[3:0] Rationale: More bits on each XOR can give better statistical uniformity on sets and since two lsbs are taken for each tile row size, it reduces the chance of aliasing on sets.	All
11	Scheme 3	New_set_mask[3] = Tiled_address[22] ^	All

		Tiled_address[21] ^ Tiled_address[20] ^ Tiled_address[19] New_set_mask[2] = Tiled_address[18] ^ Tiled_address[17] ^ Tiled_address[16] New_set_mask[1] = Tiled_address[15] ^ Tiled_address[14] New_set_mask[0] = Tiled_address[13] New_set[3:0] <= New_set_mask[3:0] ^ Old_set[3:0] Rationale: More bits on each XOR can give better statistical uniformity on sets and since each XOR has different bits, it reduces the chance of aliasing on sets even more.	

6:5	**Reserved**	Project:	DevSNB	Format:

4 **Data Cache Disable**

Project: All

Default Value: 0h

Format: Disable

Value	Name	Description	Project
0h		Cache is enable	All
1h		Reserved	All

3 **Depth Read Hit Write-Only Optimization Disable**

Project: DevSNB

Default Value: 0h

Format: Disable

Value	Name	Description	Project
0h		Read Hit Write-only optimization is enabled in the Depth cache (RCZ).	DevSNB+
1h		Read Hit Write-only optimization is disabled in the Depth cache (RCZ).	DevSNB+

Cache_Mode_1— Cache Mode Register 1

2

Depth Cache LRA Hunt Feature Disable

Project: DevSNB

Default Value: 0h

Format: Disable

Value	Name	Description	Project
0h		LRA Hunt eviction policy is enabled for Depth Cache (RCZ).	DevSNB+
1h		LRA Hunt eviction policy is disabled. In this case, strict LRA eviction policy is used in Depth Cache(RCZ).	DevSNB+

1

Instruction and State Level 2 Cache Disable

Project: All

Default Value: 0h

Format: Disable

ISC cache must be invalidated before toggling this bit.

Value	Name	Description	Project
0h		Cache is enabled.	All
1h		Reserved	All

0

Instruction Level 1 Cache Disable

Project: All

Default Value: 0h

Format: Disable

Value	Name	Description	Project
0h		Cache is enabled.	All
1h		Reserved	All

1.1.5.6 INSTPM—Instruction Parser Mode Register

INSTPM—Instruction Parser Mode Register	
Register Type:	MMIO_CS
Address Offset:	20C0h
Project:	All
Default Value:	00006000h
Access:	R/W
Size (in bits):	32
Trusted Type:	1

The INSTPM register is used to control the operation of the Instruction Parser. Certain classes of instructions can be disabled (ignored) – often useful for detecting performance bottlenecks. Also, "Synchronizing Flush" operations can be initiated – useful for ensuring the completion (vs. only parsing) of rendering instructions.

Programming Notes:

- If an instruction type is disabled, the parser will read those instructions but not process them.
- Error checking will be performed even if the instruction is ignored.
- All Reserved bits are implemented.
- This Register is saved and restored as part of Context.

Bit	Description
31:16	**Mask Bits** Format: Mask[15:0] **Masks:** These bits serve as write enables for bits 15:0. If this register is written with any of these bits clear the corresponding bit in the field 15:0 will not be modified. Reading these bits always returns 0s.
15	**Reserved** Project: All Format: MBZ
14:13	**Reserved**
12	**Reserved**
11	**CLFLUSH Toggle** Project: DevSNB+ Format: U1 This bit changes polarity each time the MI_CLFLUSH command completes.
10	**Reserved** Project: All Format: MBZ
9	**TLB Invalidate** Project: DevSNB+ Format: U1 If set, this bit allows the command stream engine to invalidate the 3D render TLBs. This bit is valid only with the Sync flush enable. ***Note: GFX soft resets do not invalidate TLBs, it is upto GFX driver to explicitly invalidate TLBs post reset.***

INSTPM—Instruction Parser Mode Register	
8	**Memory Sync Enable** Project: DevSNB+ Format: U1 If set, this bit allows the command stream engine to write out the data from the local caches to memory. This bit is valid only with the Sync flush enable
7	**Force Sync Command Ordering** Project: DevSNB+ Format: Enable By default, driver/OS synchronization commands (MI_STORE_DATA_IMM, for instance) can execute out of order with respect to 3D state and 3D primitive commands. When set, this bit forces ordering of these command. See section 3.2.2 for a list of these commands
6	**CONSTANT_BUFFER Address Offset Disable** Project: All Format: U1 When this bit is clear, the 3DSTATE_CONSTANT_* Buffers' Starting Address is used as a DynamicStateOffset. I.e., it serves as an offset from the Dynamic State Base Address **[DevSNB+]**. Accesses will be subject to Dynamic State bounds checking. When this bit is set, the 3DSTATE_CONSTANT_* Buffers' Starting Address is used as a true GraphicsAddress (not an offset). No bounds checking will be performed during access. Format = Disable
5	**Sync Flush Enable** Project: All Format: U1 This field is used to request a Sync Flush operation. The device will automatically clear this bit before completing the operation. See Sync Flush (*Programming Environment*). **Programming Note:** • The command parser must be stopped prior to issuing this command by setting the **Stop Rings** bit in register **MI_MODE**. Only after observing **Rings Idle** set in **MI_MODE** can a Sync Flush be issued by setting this bit. Once this bit becomes clear again, indicating flush complete, the command parser is re-enabled by clearing **Stop Rings**. • Sync flush in multi-context scheduling mode can be used only if there is one context in hardware and no new contexts can be scheduled till sync flush is complete. • Software is expected to follow restriction above or not use Sync flush in multi-context scheduling mode Format = Enable (cleared by HW) DevSNB{WA: D2}: If 0x21d0[7] = '1', the following work-around is needed Write 0x2054[31:0] = 0x000FFFF <-- Set the idle counter to max value Write 0x2700[31:0] = 0x00000000 <-- Wake up CS (but don't do anything) Poll 0x22AC[3:0] = 0 <-- Guarantees render pipe is awake Write 0x2050[31:0] = 0x00010001 <-- disable sequence VT-d request(Sync Flush) <-- Normal VT-d cycles(Replace with Sync Flush Steps) Write 0x2054[31:0] = <old value> <-- Set to value before flow began Write 0x2050[31:0] = 0x00010000 <-- Enable sequence (to enter RC6)
4	**Reserved** Project: All

INSTPM—Instruction Parser Mode Register	
3	**Media Instruction Disable** Project: All Format: U1 This bit instructs the Renderer instruction parser to parse and error-check Media instructions, but not execute them. Format = Disable
2	**3D Rendering Instruction Disable** Project: All Format: U1 This bit instructs the Renderer instruction parser to parse and error-check 3D Rendering instructions, but not execute them. This bit must always be set by software if **3D State Instruction Disable** is set. Setting this bit *without* setting **3D State Instruction Disable** *is* allowed. Format = Disable
1	**3D State Instruction Disable** Project: All Format: U1 This bit instructs the Renderer instruction parser to parse and error-check 3D State instructions, but not execute them. This bit should *not* be set unless **3D Rendering Instruction Disable** (bit 2) is also set. Format = Disable
0	**Texture Palette Load Instruction Disable** Project: All Format: U1 This bit instructs the Renderer instruction parser to parse and error-check Texture Palette Load instructions, but not execute them. Format = Disable

1.1.5.7 EXCC—Execute Condition Code Register

EXCC—Execute Condition Code Register	
Register Type:	MMIO_CS
Address Offset:	2028h
Project:	All
Default Value:	00000000h
Access:	R/W,RO
Size (in bits):	32
Trusted Type:	1

This register contains user defined and hardware generated conditions that are used by MI_WAIT_FOR_EVENT commands. An MI_WAIT_FOR_EVENT instruction excludes the executing ring from arbitration if the selected event evaluates to a “1”, while instruction is discarded if the condition evaluates to a “0”. Once excluded a ring is enabled into arbitration when the selected condition evaluates to a “0”.
This register also contains control for the invalidation of indirect state pointers on context restore.

Bit	Description
31:16	**Mask Bits** Format: Mask[15:0] These bits serves as a write enable for bits 15:0. If this register is written with any of these bits clear the corresponding bit in the field 15:0 will not be modified. Reading these bits always returns 0s.
15:12	**Reserved** Project: All Format: MBZ
11	**Pending Indirect State Dirty Bit** Project: All Format: U32 This field keeps track of whether or not an indirect state pointer command has been parsed in the current context. Clears either on a context save or explicitly through a flush command
10:7	**Pending Indirect State Counter** Project: All Format: U32 This field keeps track of the maximum number of indirect state pointers pending in the system. When the register is saved/restored, it saves either a value of 1 or 0. This field is Read-Only
6:5	**Reserved** Project: All Format: MBZ
4:0	**User Defined Condition Codes** The software may signal a Stream Semaphore by setting the Mask bit and Signal Bit together to match the bit field specified in a WAIT_FOR_EVENT (Semaphore).

1.1.5.8 FBC RT BASE ADDRESS REGISTER

FBC_RT_BASE_ADDR_REGISTER	
Register Type:	MMIO
Address Offset:	2128h {DevSNB]
Project:	All
Default Value:	--
Access:	Read/32 bit Write
Size (in bits):	32
This Register is saved and restored as part of Context.	

Bit	Description
31:12	4KB aligned Base Address as mapped in the PPGTT (in the multi-context scheduling mode) OR in the GGTT (in the single-context scheduling mode) For the render target. This register must be programmed in either multi-context scheduling or single-context scheduling mode. This base address must be the one that is either front buffer or the back-buffer (a flip target). It can be only programmed once per context. It must be programmed before any draw call binding that render target base address. Format: Base Address[31:12] Must be set to modify corresponding data bit. Reads to this field returns zero.
11:2	**Reserved** Project: All Format: MBZ
1	**FBC Front Buffer Target** Project: All Default Value: 0h Format: Enable

Value	Name	Description	Project
0h		FBC is targeting the Back Buffer for compression. This buffer **can** be cached in the MLC/LLC, so a GFDT flush is required before FBC can begin compression.	All
1h		FBC is targeting the Font Buffer for compression. This buffer **cannot** be cached in the MLC/LLC. FBC compression can begin after any RC flush.	All

FBC_RT_BASE_ADDR_REGISTER	
0	**PPGTT Render Target Base Address Valid for FBC** Project: All Security: None Access: None Exists If: Always Default Value: 0h DefaultVaueDesc Mask: MMIO(0x2000)#16 Format: Enable FormatDesc Address: GraphicsAddress[31:0] Surface Type: U32 Range 0..2^32-1 BitFieldDesc

Value	Name	Description	Project
0h		Base address in this register [31:12] is not valid and therefore FBC will not get any modifications from rendering.	
1h		Base address in this register [31:12] is valid and HW needs to compare the current render target base address with this base address to provide modifications to FBC.	

1.1.5.9 RVSYNC – Render/Video Semaphore Sync Register

RVSYNC – Render/Video Semaphore Sync Register

Register Type: MMIO_CS
Address Offset: 2040h
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 32
Trusted Type: 1

This register is written by VCS, read by CS.

Bit	Description
31:0	**Semaphore Data** Semaphore data for synchronization between render engine and video codec engine.

1.1.5.10 RBSYNC – Render/Blitter Semaphore Sync Register

RBSYNC – Render/Blitter Semaphore Sync Register	
Register Type:	MMIO_CS
Address Offset:	2044h
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Trusted Type:	1

This register is written by BCS, read by CS.

Bit	Description
31:0	**Semaphore Data** Semaphore data for synchronization between render engine and blitter engine.

1.1.5.11 SEMA_REG—Semaphore General Sync Registers

SEMA_REG—Semaphore General Sync Registers	
Register Type:	MMIO_CS
Address Offset:	2680-26FFh
Project:	All
Default Value:	0h
Access:	R/W
Size (in bits):	32 registers x 32b

This register contains the semaphore value to be compared with the value specifed in the MI_SEMAPHORE_MBOX command. The register value in the command will be compared with the MMIO offset specifed in the table below:

Register Number	MMIO Offset
0	0x2680
1	0x2684
2	0x2688
3	0x268C

SEMA_REG—Semaphore General Sync Registers		
4	0x2690	
5	0x2694	
6	0x2698	
7	0x269C	
8	0x26A0	
9	0x26A4	
10	0x26A8	
11	0x26AC	
12	0x26B0	
13	0x26B4	
14	0x26B8	
15	0x26BC	
16	0x26C0	
17	0x26C4	
18	0x26C8	
19	0x26CC	
20	0x26D0	
21	0x26D4	
22	0x26D8	
23	0x26DC	

SEMA_REG—Semaphore General Sync Registers

24	0x26E0	
25	0x26E4	
26	0x26E8	
27	0x26EC	
28	0x26F0	
29	0x26F4	
30	0x26F8	
31	0x26FC	

Bit	Description
31:0	**Semaphore Data** Semaphore data for synchronization between render engine and video codec engine.

1.1.6 RINGBUF — Ring Buffer Registers

See the “Device Programming Environment” chapter for detailed information on these registers

1.1.6.1 RING_BUFFER_TAIL

RING_BUFFER_TAIL	
Register Type:	MMIO_CS
Address Offset:	2030h
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

These registers are used to define and operate the “ring buffer” mechanism which can be used to pass instructions to the command interface. The buffer itself is located in a physical memory region. The ring buffer is defined by a 4 Dword register set that includes starting address, length, head offset, tail offset, and control information. Refer to the *Programming Interface* chapter for a detailed description of the parameters specified in this ring buffer register set, restrictions on the placement of ring buffer memory, arbitration rules, and in how the ring buffer can be used to pass instructions.

Ring Buffer Tail Offsets must be properly programmed before ring is enabled. A Ring Buffer can be enabled when empty.

Bit	Description
31:21	**Reserved** Project: All Format: MBZ
20:3	**Tail Offset** Project: All Format: U18 QWord Offset This field is written by software to specify where the valid instructions placed in the ring buffer end. The value written points to the QWord *past* the last valid QWord of instructions. In other words, it can be defined as the *next* QWord that software will write instructions into. Software must write subsequent instructions to QWords following the Tail Offset, possibly wrapping around to the top of the buffer (i.e., software can’t skip around within the buffer). Note that all DWords prior to the location indicated by the **Tail Offset** must contain valid instruction data – which may require instruction padding by software. See **Head Offset** for more information.
2:0	**Reserved** Project: All Format: MBZ

1.1.6.2 RING_BUFFER_HEAD

RING_BUFFER_HEAD	
Register Type:	MMIO_CS
Address Offset:	2034h
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

These registers are used to define and operate the "ring buffer" mechanism which can be used to pass instructions to the command interface. The buffer itself is located in a physical memory region. The ring buffer is defined by a 4 Dword register set that includes starting address, length, head offset, tail offset, and control information. Refer to the *Programming Interface* chapter for a detailed description of the parameters specified in this ring buffer register set, restrictions on the placement of ring buffer memory, arbitration rules, and in how the ring buffer can be used to pass instructions.

Ring Buffer Head Offsets must be properly programmed before ring is enabled. A Ring Buffer can be enabled when empty.

Bit	Description
31:21	**Wrap Count** Project: All Default Value: 0h Format: U11 — count of ring buffer wraps This field is incremented by 1 whenever the **Head Offset** wraps from the end of the buffer back to the start (i.e., whenever it wraps back to 0). Appending this field to the **Head Offset** field effectively creates a virtual 4GB Head "Pointer" which can be used as a tag associated with instructions placed in a ring buffer. The Wrap Count itself will wrap to 0 upon overflow.

RING_BUFFER_HEAD

20:2	**Head Offset** Project: All Format: U19 DWord Offset This field indicates the offset of the *next* instruction DWord to be parsed. Software will initialize this field to select the first DWord to be parsed once the RB is enabled. (Writing the Head Offset while the RB is enabled is UNDEFINED). Subsequently, the device will increment this offset as it executes instructions – until it reaches the QWord specified by the **Tail Offset**. At this point the ring buffer is considered "empty". **Programming Notes** — **Project** A RB can be enabled empty or containing some number of valid instructions. — All
1	**Reserved** Project: All Format: MBZ
0	**Wait for Condition Indicator** Project: All Format: Enabled This is a read only value used to indicate whether or not the command streamer is currently waiting for a conditional code to be cleared from 0x2028

1.1.6.3 RING_BUFFER_START

RING_BUFFER_START	
Register Type:	MMIO_CS
Address Offset:	2038h
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

These registers are used to define and operate the "ring buffer" mechanism which can be used to pass instructions to the command interface. The buffer itself is located in a physical memory region. The ring buffer is defined by a 4 Dword register set that includes starting address, length, head offset, tail offset, and control information. Refer to the *Programming Interface* chapter for a detailed description of the parameters specified in this ring buffer register set, restrictions on the placement of ring buffer memory, arbitration rules, and in how the ring buffer can be used to pass instructions.

Bit	Description
31:12	**Starting Address** Project: All Address: GraphicsAddress[31:12] Surface Type: RingBuffer This field specifies Bits 31:12 of the 4KB-aligned starting Graphics Address of the ring buffer. Address bits 31 down to 29 must be zero. All ring buffer pages must map to Main Memory (uncached) pages. Ring Buffer addresses are always translated through the global GTT.
11:0	**Reserved** Project: All Format: MBZ

1.1.6.4 RING_BUFFER_CONTROL

RING_BUFFER_CONTROL	
Register Type:	MMIO_CS
Address Offset:	203Ch
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

These registers are used to define and operate the "ring buffer" mechanism which can be used to pass instructions to the command interface. The buffer itself is located in a physical memory region. The ring buffer is defined by a 4 Dword register set that includes starting address, length, head offset, tail offset, and control information. Refer to the *Programming Interface* chapter for a detailed description of the parameters specified in this ring buffer register set, restrictions on the placement of ring buffer memory, arbitration rules, and in how the ring buffer can be used to pass instructions.

Bit	Description
31:21	**Reserved** Project: All Format: MBZ
20:12	**Buffer Length** Project: All Format: U9 Count of 4 KB pages Range 0..1FF This field is written by SW to specify the length of the ring buffer in 4 KB Pages. Range = [0 = 1 page = 4 KB, 1FFh = 512 pages = 2 MB]
11	**RB Wait** Project: All Format: Boolean Indicates that this ring has executed a WAIT_FOR_EVENT instruction and is currently waiting. Software can write a "1" to clear this bit, write of "0" has no effect. When the RB is waiting for an event and this bit is cleared, the wait will be terminated and the RB will be returned to arbitration.
10	**Semaphore Wait** Project: DevSN B+ Format: U32 Indicates that this ring has executed a MI_SEMAPHORE_MBOX instruction with register compare and is currently waiting. Software can write a "1" to clear this bit, write of "0" has no effect. When the RB is waiting for the compare to meet and this bit is cleared, the wait will be terminated and the RB will be returned to arbitration.
9:3	**Reserved** Project: All Format: MBZ

RING_BUFFER_CONTROL

2:1 Automatic Report Head Pointer

Project: All

This field is written by software to control the automatic "reporting" (write) of this ring buffer's "Head Pointer" register (register DWord 1) to the corresponding location within the Hardware Status Page. Automatic reporting can either be disabled or enabled at 4KB, 64KB or 128KB boundaries within the ring buffer.

Value	Name	Description	Project
0h	MI_AUTOREPORT_OFF	Automatic reporting disabled	All
1h	MI_AUTOREPORT_64KB MI_AUTOREPORT_4KB	Report every 16 pages (64KB) When the **Per-Process Virtual Address Space** bit is set, the ring buffer reports every 4KB	All
2h	Reserved	Reserved	All
3h	MI_AUTOREPORT_128KB	Report every 32 pages (128KB)	All

Programming Notes	Project
When the **Per-Process Virtual Address Space** bit is set and automatic head reporting is desired, this field must be set to option 1 since the ring buffer will be only 16KB in size. The head pointer will be reported to the head pointer location in the PP HW Status Page when it passes each 4KB page boundary. When the above-mentioned bit is reset, reporting will behave just as on the prior devices (as documented above), and option 1 will report on 64KB boundary.	All

0 Ring Buffer Enable Project: All Format: Enable

This field is used to enable or disable this ring buffer. It can be enabled or disabled regardless of whether there are valid instructions pending. If disabled and the ring head equals ring tail, all state currently loaded in hardware is considered *invalid*.

1.1.6.5 UHPTR — Pending Head Pointer Register

UHPTR — Pending Head Pointer Register

Register Type: MMIO_CS
Address Offset: 2134h
Project: All
Default Value: 0000 0000h
Access: R/W
Size (in bits): 32

Bit	Description
31:3	**Head Pointer Address** Project: All Default Value: 0h Address: GraphicsAddress[31:3] This register represents the GFX address offset where execution should continue in the ring buffer following execution of an MI_ARB_CHECK command.
2:1	**Reserved** Project: All Format: MBZ
0	**Head Pointer Valid** Project: All Default Value: 0h Format: U1 This bit is set by the software to request a pre-emption. It is reset by hardware when an MI_ARB_CHECK command is parsed by the command streamer. The hardware uses the head pointer programmed in this register at the time the reset is generated.

Value	Name	Description	Project
0h		No valid updated head pointer register, resume execution at the current location in the ring buffer	All
1h		Indicates that there is an updated head pointer programmed in this register	All

1.1.7 Watchdog Timer Registers

These 2 registers together implement a watchdog timer. Writing ones to the control register enables the counter, and writing zeroes disables the counter. The 2nd register is programmed with a threshold value which, when reached, signals an interrupt then resets the counter to 0. Program the threshold value before enabling the counter or extremely frequent interrupts may result.

Note that the counter itself is not observable. It increments with the main render clock.

1.1.7.1 PR_CTR_CTL—Render Watchdog Counter Control

PR_CTR_CTL—Render Watchdog Counter Control	
Register Type:	MMIO_CS
Address Offset:	2178h
Project:	All
Default Value:	0000 0001h
Access:	R/W
Size (in bits):	32

Bit	Description
31	**Count Select** Project: [DevSNB] Format: select 0 – Use the timestamp to increment the watchdog count (every 640ns) 1 – Use the fixed function clock (csclk) to increment the watchdog count
30:0	**Counter logic op** Project: All Format: U32 This field specifies the action to be taken by the clock counter to generate interrupts. Writing 0 into this register causes a core render clock counter to be kicked off. Writing 1 into this register causes a core render clock counter to be stopped and reset to 0.

1.1.7.2 PR_CTR_THRSH—Render Watchdog Counter Threshold

PR_CTR_THRSH—Render Watchdog Counter Threshold	
Register Type:	MMIO_CS
Address Offset:	217Ch
Project:	All
Default Value:	0014 5855h
Access:	R/W
Size (in bits):	32

Bit	Description
31:0	**Counter logic Threshold** Project: All Format: U32 This field specifies the threshold that the hardware checks against for the value of the render clock counter before generating an interrupt. The counter in hardware generates an interrupt when the threshold is reached, rolls over and starts counting again. The interrupt generated is the "Media Hang Notify" interrupt since this watchdog timer is intended primarily to remedy VLD hangs on the main pipeline.

1.1.7.3 PR_CTR—Render Watchdog Counter

PR_CTR—Render Watchdog Counter	
Register Type:	MMIO_CS
Address Offset:	2190h
Project:	All
Default Value:	0000 0000h
Access:	RO
Size (in bits):	32

Bit	Description
31:0	**Counter Value** Project: All Format: U32 This register reflects the render watchdog counter value itself. It cannot be written to.

1.1.8 Interrupt Control Registers

The Interrupt Control Registers described below all share the same bit definition. The bit definition is as follows:

Table 1-1. Bit Definition for Interrupt Control Registers

Bit	Description
31:10	**Reserved. MBZ** These bits may be assigned to interrupts on future products/steppings.
9	**Performance Monitoring Buffer Half-Full Interrupt:** For internal trigger (timer based) based reporting, if the report buffer crosses half full limit, this interrupt is generated.
8	**Context Switch Interrupt:** Set when a context switch has just occurred. **Per-Process Virtual Address Space bit** needs to be set for this interrupt to occur**.**
7	**Page Fault:** This bit is set whenever there is a pending PPGTT (page or directory) fault.
6	**Timeout Counter Expired:** Set when the render pipe timeout counter (0x02190) has reached the timeout thresh-hold value (0x0217c).
5	**L3 Parity Error:** When this bit is set, L3 cache controller is indicating that it has encountered an parity error while checking the data.
4	**PIPE_CONTROL Notify Interrupt:** The Pipe Control packet (Fences) specified in *3D pipeline* document may optionally generate an Interrupt. The Store QW associated with a fence is completed ahead of the interrupt.
3	**Render Command Parser Master Error:** When this status bit is set, it indicates that the hardware has detected an error. It is set by the device upon an error condition and cleared by a CPU write of a one to the appropriate bit contained in the Error ID register followed by a write of a one to this bit in the IIR. Further information on the source of the error comes from the "Error Status Register" which along with the "Error Mask Register" determine which error conditions will cause the error status bit to be set and the interrupt to occur. **Page Table Error:** Indicates a page table error. **Instruction Parser Error**: The Renderer Instruction Parser encounters an error while parsing an instruction.
2	**Sync Status:** This bit is set in the Hardware Status Page DW offset 0 when the Instruction Parser completes a flush with the sync enable bit active in the INSTPM register. The toggle event will happen after the render engine is flushed. The HW Status DWord write resulting from this toggle will cause the CPU's view of graphics memory to be coherent as well (flush and invalidate the render cache). **It is the driver's responsibility to clear this bit before the next sync flush with HWSP write enabled**
1	**Reserved**
0	**Render Command Parser User Interrupt:** This status bit is set when an MI_USER_INTERRUPT instruction is executed on the Render Command Parser. Note that instruction execution is not halted and proceeds normally. A mechanism such as an MI_STORE_DATA instruction is required to associate a particular meaning to a user interrupt.

The following table specifies the settings of interrupt bits stored upon a “Hardware Status Write” due to ISR changes:

Bit	Interrupt Bit	ISR bit Reporting via Hardware Status Write (when unmasked via HWSTAM)
9	**Performance Monitoring Buffer Half-Full Interrupt**	Set when event occurs, cleared when event cleared
8	**Context Switch Interrupt:** Set when a context switch has just occurred.	Not supported to be unmasked
7	**Page Fault:** This bit is set whenever there is a pending PPGTT (page or directory) fault.	Set when event occurs, cleared when event cleared
6	**Reserved**	
5	**Reserved**	
4	PIPE_CONTROL packet - Notify Enable	0
3	Master Error	Set when error occurs, cleared when error cleared
2	Sync Status	Toggled every SyncFlush Event
1	Reserved	
0	User Interrupt	0

1.1.8.1 HWSTAM — Hardware Status Mask Register

Hardware Status Mask Register	
Register Type:	MMIO_CS
Address Offset:	2098h
Project:	All
Default Value:	**FFFF FFFFh**
Access:	R/W, RO
Size (in bits):	32
Trusted Type:	1

The HWSTAM register has the same format as the Interrupt Control Registers. The bits in this register are “mask” bits that prevent the corresponding bits in the Interrupt Status Register from generating a “Hardware Status Write” (PCI write cycle). Any unmasked interrupt bit (HWSTAM bit set to 0) will allow the Interrupt Status Register to be written to the ISR location (within the memory page specified by the Hardware Status Page Address Register) when that Interrupt Status Register bit changes state.

Programming Note:

- To write the interrupt to the HWSP, the corresponding IMR bit must also be clear (enabled).
- At most 1 bit can be unmasked at any given time.

Bit	Description
31:0	**Hardware Status Mask Register** Project: All Default Value: FFFFFFFFh DefaultVaueDesc Format: Array of Masks Refer to Interrupt Control Register section for bit definitions, Reserved bits are RO

1.1.8.2 IMR—Interrupt Mask Register

IMR—Interrupt Mask Register

Register Type: MMIO_CS
Address Offset: 20A8h
Project: All
Default Value: FFFF FFFFh
Access: R/W, RO
Size (in bits): 32

The IMR register is used by software to control which Interrupt Status Register bits are “masked” or “unmasked”. “Unmasked” bits will be reported in the IIR, possibly triggering a CPU interrupt, and will persist in the IIR until cleared by software. “Masked” bits will not be reported in the IIR and therefore cannot generate CPU interrupts.

Bit	Description
31:0	**Interrupt Mask Bits** Project: All Default Value: FFFF FFFFh Format: Array of interrupt mask bits — Refer to Table 3-4 in Interrupt Control Register section for bit definitions This field contains a bit mask which selects which interrupt bits (from the ISR) are reported in the IIR. Reserved biths in teh Interrupt Control Register are RO

Value	Name	Description	Project
0h	Not Masked	Will be reported in the IIR	All
1h	Masked	Will not be reported in the IIR	All

1.1.8.3 Hardware-Detected Error Bit Definitions (for EIR, EMR, ESR)

This section defines the Hardware-Detected Error bit definitions and ordering that is common to the EIR, EMR and ESR registers. The EMR selects which error conditions (bits) in the ESR are reported in the EIR. Any bit set in the EIR will cause the Master Error bit in the ISR to be set. EIR bits will remain set until the appropriate bit(s) in the EIR is cleared by writing the appropriate EIR bits with '1' (except for the unrecoverable bits described below).

The following table describes the Hardware-Detected Error bits:

Table 1-2. Hardware-Detected Error Bits

Bit	Description
31:5	Reserved: MBZ
4	**Page Table Error**: This bit is set when a Graphics Memory Mapping Error is detected. The cause of the error is indicated (to some extent) in the PGTBL_ER register. Note: This error indications can not be cleared except by reset (i.e., it is a fatal error). 1 = Page table error
3	**Memory Privilege Violation Error.** This bit is set if a command in a non-secure batch buffer attempts an operation to the GGTT (this can only happen in commands that contain a PPGTT vs. GGTT selector). The command will be executed as if the selector bit indicated PPGTT and parsing will continue.
2	**Command Privilege Violation Error.** This bit is set if a command classified as privileged is parsed in a non-secure batch buffer. The command will be converted to a NOOP and parsing will continue.
1	Reserved: MBZ
0	**Instruction Error:** This bit is set when the Renderer Instruction Parser detects an error while parsing an instruction. Instruction errors include: 1) Client ID value (Bits 31:29 of the Header) is not supported (only MI, 2D and 3D are supported). 2) Defeatured MI Instruction Opcodes: 1: Instruction Error detected. Note: This error indications cannot be cleared except by reset (i.e., it is a fatal error).

1.1.8.3.1 EIR — Error Identity Register

EIR — Error Identity Register

Register Type: MMIO_CS
Address Offset: 20B0h
Project: All
Default Value: 0000 0000h
Access: R/W, RO
Size (in bits): 32

The EIR register contains the persistent values of Hardware-Detected Error Condition bits. Any bit set in this register will cause the Master Error bit in the ISR to be set. The EIR register is also used by software to clear detected errors (by writing a '1' to the appropriate bit(s)), except for ther unrecoverable bits described.)

Bit	Description
31:16	**Reserved** Project: All Format: MBZ
15:0	**Error Identity Bits** Project: All Default Value: 0h Format: Array of Error condition bits See Table 1 5. Hardware-Detected Error Bits This register contains the persistent values of ESR error status bits that are unmasked via the EMR register. The logical OR of all (defined) bits in this register is reported in the Master Error bit of the Interrupt Status Register. In order to clear an error condition, software must first clear the error by writing a '1' to the appropriate bit(s) in this field. If required, software should then proceed to clear the Master Error bit of the IIR. Reserved bits are RO.

Bit	Description
31:5	Reserved: MBZ

Value	Name	Description	Project
1h	Error occurred	Error occurred	All

Programming Notes	Project
Writing a '1' to a set bit will cause that error condition to be cleared. However, the Page Table Error bit (Bit 4) nor the Instruction Error bit (Bit 0) cannot be cleared except by reset (i.e., it is a fatal error).	All

1.1.8.3.2 EMR—Error Mask Register

EMR—Error Mask Register

Register Type: MMIO_CS
Address Offset: 20B4h
Project: All
Default Value: FFFF FFFFh
Access: R/W, RO
Size (in bits): 32

The EMR register is used by software to control which Error Status Register bits are "masked" or "unmasked". "Unmasked" bits will be reported in the EIR, thus setting the Master Error ISR bit and possibly triggering a CPU interrupt, and will persist in the EIR until cleared by software. "Masked" bits will not be reported in the EIR and therefore cannot generate Master Error conditions or CPU interrupts. Reserved bits are RO.

Bit	Description
31:16	**Reserved** Project: All Format: MBZ
15:0	**Error Mask Bits** Project: All Default Value: FFFF FFDFh Format: Array of error condition mask bits See Table 1 5. Hardware-Detected Error Bits This register contains a bit mask that selects which error condition bits (from the ESR) are reported in the EIR.

Value	Name	Description	Project
0h	Not Masked	Will be reported in the EIR	All
1h	Masked	Will not be reported in the EIR	All

1.1.8.3.3 ESR—Error Status Register

ESR—Error Status Register

Register Type: MMIO_CS
Address Offset: 20B8h
Project: All
Default Value: 0000 0000h
Access: RO
Size (in bits): 32

The ESR register contains the current values of all Hardware-Detected Error condition bits (these are all by definition "persistent"). The EMR register selects which of these error conditions are reported in the persistent EIR (i.e., set bits must be cleared by software) and thereby causing a Master Error interrupt condition to be reported in the ISR.

Bit	Description
31:16	**Reserved** Project: All Format: MBZ

ESR—Error Status Register

15:0	**Error Status Bits** Project: All Default Value: 0h Format: Array of error condition bits — See Table 1 5. Hardware-Detected Error Bits This register contains the non-persistent values of all hardware-detected error condition bits.

Value	Name	Description	Project
1h	Error Condition Detected	Error Condition detected	All

1.1.9 Logical Context Support

1.1.9.1 BB_ADDR—Batch Buffer Head Pointer Register

BB_ADDR—Batch Buffer Head Pointer Register

Register Type: MMIO_CS
Address Offset: 2140h
Project: All
Default Value: 0000 0000 0000 0000h
Access: RO
Size (in bits): 32

This register contains the current DWord Graphics Memory Address of the last-initiated batch buffer.

Programming Restriction:
This register should NEVER be programmed by driver, this is for HW internal use only.

Bit	Description
31:2	**Batch Buffer Head Pointer** Project: All Format: GraphicsAddress[31:2] This field specifies the DWord-aligned Graphics Memory Address where the last initiated Batch Buffer is currently fetching commands. If no batch buffer is currently active, the Valid bit will be 0 and this field will be meaningless.
1	**Reserved** Project: All Format: MBZ
0	**Valid** Project: All Default Value: 0h Format: U1

Value	Name	Description	Project
0h	Invalid	Batch buffer Invalid	All
1h	Valid	Batch buffer Valid	All

1.1.9.2 BB_STATE – Batch Buffer State Register

BB_STATE – Batch Buffer State Register

Register Type: MMIO_CS
Address Offset: 2110h
Project: All
Default Value: 0000 0000h
Access: RO
Size (in bits): 32

This register contains the attributes of the last batch buffer initiated from the Ring Buffer. These include the security indicator.

This register should *not* be written by software directly. Software should always set these fields via the MI_BATCH_BUFFER_START command when initiating a batch buffer.

{DevSNB] This register is *not* restored with context. As a consequence, MI_WAIT_FOR_EVENT cannot enter RC6 inside a batch buffer with any of these attributes set

Bit	Description
31:8	**Reserved** Project: All Format: MBZ
7	**Reserved** Project: Format:
6	**Clear Command Buffer Enable** Project: All Format: U1 If set the batch buffer is getting executed from the Write Once protected memory area. The address of the batch buffer is an offset into the WOPCM area.
5	**Buffer Security Indicator** Project: All Default Value: 0h Format: MI_BufferSecurityType If set, this batch buffer is non-secure and cannot execute privileged commands nor access privileged (GGTT) memory. It will be accessed via the PPGTT. If clear, this batch buffer is secure and will be accessed via the GGTT. Note: This field reflects the effective security level and may not be the same as the Buffer Security Indicator written using MI_BATCH_BUFFER_START.

Value	Name	Description	Project
0h	MIBUFFER_SECURE	Located in GGTT memory	All
1h	MIBUFFER_NONSECURE	Located in PPGTT memory	All

4	**Reserved**
3:0	**Reserved** Project: All Format: MBZ

1.1.9.3 CTXT_SR_CTL – Context Save/Restore Control Register

CTXT_SR_CTL – Context Save/Restore Control Register	
Register Type:	2714h [DevSNB]
Address Offset:	2714h
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32
This register is saved and restored with context.	

Bit	Description
31:2	**Reserved** Project: All Format: MBZ
1	**Reserved**
0	**Render Context Restore Inhibit** Project: All Format: U1 This is not a true register bit. This bit should be set in the context image of a ring context that is being submitted for the first time. Setting this bit will inhibit the restoring of render context (including extended context if applicable) so that restoring of an uninitialized render context can be prevented. This bit will always be set on a context save (since the render context cannot be uninitialized on context save – it will always contain at least default values.)

1.1.9.4 CCID—Current Context Register

CCID—Current Context Register	
Register Type:	MMIO_CS
Address Offset:	2180h
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

This register contains the current "logical rendering context address" associated with the ring buffer.

Programming Note: The CCID register must not be written directly (via MMIO) unless the Command Streamer is completely idle (i.e., the Ring Buffer is empty and the pipeline is idle). Note that, under normal conditions, the CCID register should only be updated from the command stream using the MI_SET_CONTEXT command.

CCID—Current Context Register

Bit	Description
31:12	**Logical Render Context Address (LRCA)** Project: All Default Value: 0h Address: GraphicsAddress[31:11] This field contains the 4 KB-aligned Graphics Memory Address of the current Logical Rendering Context. Bit 11 MBZ. This register will point to a Logical Pipeline Context (a subset of a Logical Rendering Context) if loaded using MI_SET_CONTEXT.
11:10	**Reserved** Project: All Format: MBZ
9	**Reserved**
8	**Reserved** Project: All Format: Must be '1'
7:4	**Reserved** Project: All Format: MBZ
3	**Extended State Save Enable** Project: All Format: Enable If set, the extended state identified in the Logical Context Data section of the Memory Data Formats chapter, is saved as part of switching away from this logical context.
2	**Extended State Restore Enable** Project: All Format: Enable If set, the extended state identified in the Logical Context Data section of the Memory Data Formats chapter, was loaded (or restored) as part of switching to this logical context.
0	**Valid** Project: All Default Value: 0h Format: U1

Value	Name	Description	Project
0h	Invalid	The other fields of this register are invalid. A switch away from the context will not invoke a context save operation.	All
1h	Valid	The other fields of this register are valid, and a switch from the context will invoke the normal context save/restore operations.	All

1.1.9.5 CXT_SIZE—Context Sizes

CXT_SIZE—Context Sizes

Register Type:	MMIO_CS
Address Offset:	Write: 21A8h, Read: 21A0h
Project:	DevSNB
Default Value:	1E0CDDD3h
Access:	Read/32 bit Write
Size (in bits):	32

Bit	Description
31:30	**Reserved** Project: All Format: MBZ
29:24	**Power Context Size** Project: Dev SNB Default Value: 1Eh DefaultVaueDesc Format: U32 FormatDesc BitFieldDesc
23:18	**Ring Context Size** Project: Dev SNB Default Value: 3h DefaultVaueDesc Format: U32 FormatDesc BitFieldDesc
17:12	**Render Context Size** Project: Dev SNB Default Value: Dh DefaultVaueDesc Format: U32 FormatDesc BitFieldDesc
11:6	**Extended Context Size** Project: Dev SNB Default Value: 37h DefaultVaueDesc Format: U32 FormatDesc BitFieldDesc
5:0	**3D Pipeline State Context Size** Project: Dev SNB Default Value: 13h DefaultVaueDesc Format: U32 FormatDesc BitFieldDesc

1.1.9.6 CXT_PIPESTATEBASE — Pipeline State Base Address

CXT_PIPESTATEBASE — Pipeline State Base Address	
Register Type:	MMIO_CS
Address Offset:	21B0h
Project:	DevSNB
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
This register contains the base address where the pipeline state data is saved when PSMI interruption granularity in GFX_MODE is set to mid-triangle	
Bit	**Description**
31:12	**Pipeline State Base Address** Project: All Default Value: 0h Invalid base address Format: Address Page Base Address The page aligned base address for pipelined state context data. **Programming Notes** • There must be 4 contiguous pages allocated with this base address to support 8 pipeline state specific context data
11:1	**Reserved** Project: All Format: MBZ
0	**Valid** Project: All Format: Bool Valid bit for 31:12. Defaults to invalid (clear)

1.1.9.7 MTCH_CID_RST – Matched Context ID Reset Register

MTCH_CID_RST – Matched Context ID Reset Register

Register Type:	MMIO_CS
Address Offset:	2524h
Project:	All
Default Value:	0000 0002h
Access:	R/W
Size (in bits):	32

This register is used to generate a Context ID specific reset (Render Only). To initiate a reset, the register is written with the pending bit set. Hardware compares the current context ID with the register and on match generates a Render Only reset. After reset is complete, HW clears the pending bit and can be programmed to generate an interrupt. The match bit is set. If the current context ID does not match this register, the pending bit is reset and an interrupt is generated. The match bit is reset.

The match indicates the result of the last comparison, and its valid only when pending bit is zero.

Please see MCIDRST interrupt bit assignment in the Interrupt Control Registers.

Bit	Description
31:12	**Match Context ID** Project: All Format: U20 Contains the context ID to be compared with the currently running context ID.
11:2	**Reserved** Project: All Format: MBZ
1	**Match** Project: All Format: U20 This bit indicates the result of the match operation; 1 means the Current Context ID matches the Match Context ID field.
0	**Pending** Project: All Format: U20 This bit indicates that a matched context ID reset is pending. The bit should be set when the register is written (in order to have a pending MTCH_CID_RST request), and will be reset by hardware to indicate that the operation is completed (Either with a match or mismatch)

1.1.9.8 SYNC_FLIP_STATUS – Wait for event and Display flip flags Register

SYNC_FLIP_STATUS – Wait for event and Display flip flags Register

Register Type: MMIO_CS
Address Offset: 25A0h

Project: All
Default Value: 0000 0000h
Access: R/W
Size (in bits): 32

This register is the saved value of what wait for events are still valid. This register is part of context save and restore for RC6 feature..

Bit	Description
31	**Reserved** Project: All Format: MBZ
30	**Display Plane A Asyncronous Display Flip Pending** Project: All Format: Enable This field enables a wait for the duration of a Display Plane A "Flip Pending" condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of *MI Functions*.
29	**Display Plane A Syncronous Flip Display Pending** Project: All Format: Enable This field enables a wait for the duration of a Display Plane A "Flip Pending" condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of *MI Functions*.
28	**Display Sprite A Syncronous Flip Display Pending** Project: All Format: Enable This field enables a wait for the duration of a Display Sprite A "Flip Pending" condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of *MI Functions*.
27	**Reserved** Project: All Format: MBZ

SYNC_FLIP_STATUS – Wait for event and Display flip flags Register					
26	**Display Plane B Asyncronous Display Flip Pending**	Project:	All	Format:	Enable
	This field enables a wait for the duration of a Display Plane B “Flip Pending” condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of *MI Functions*.				
25	**Display Plane B Syncronous Flip Display Pending**	Project:	All	Format:	Enable
	This field enables a wait for the duration of a Display Plane B “Flip Pending” condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of *MI Functions*.				
24	**Display Sprite B Syncronous Flip Display Pending**	Project:	All	Format:	Enable
	This field enables a wait for the duration of a Display Sprite B “Flip Pending” condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of *MI Functions*.				
23	**Display Plane A Asyncronous Performance Flip Pending Wait Enable**	Project:	All	Format:	Enable
	This field enables a wait for the duration of a Display Plane A “Flip Pending” condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of *MI Functions*.				
22	**Display Plane A Asyncronous Flip Pending Wait Enable**	Project:	All	Format:	Enable
	This field enables a wait for the duration of a Display Plane A “Flip Pending” condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of *MI Functions*.				
21	**Display Plane A Syncronous Flip Pending Wait Enable**	Project:	All	Format:	Enable
	This field enables a wait for the duration of a Display Plane A “Flip Pending” condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of *MI Functions*.				

SYNC_FLIP_STATUS – Wait for event and Display flip flags Register	
20	**Display Sprite A Syncronous Flip Pending Wait Enable** Project: All Format: Enable This field enables a wait for the duration of a Display Sprite A “Flip Pending” condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of *MI Functions*.
19	**Reserved** Project: All Format: MBZ
18	**Display Pipe A Scan Line Wait Enable** Project: All Format: Enable This field enables a wait while a Display Pipe A “Scan Line” condition exists. This condition is defined as the the start of the scan line specified in the Pipe A Display Scan Line Count Range Compare Register. See Scan Line Event in the Device Programming Interface chapter of *MI Functions*.
17	**Display Pipe A Vertical Blank Wait Enable** Project: All Format: U32 This field enables a wait until the next Display Pipe A “Vertical Blank” event occurs. This event is defined as the start of the next Display Pipe A vertical blank period. Note that this can cause a wait for up to an entire refresh period. See Vertical Blank Event (See *Programming Interface*).
16	**Display Pipe A H Blank Wait Enable** Project: All Format: Enable This field enables a wait until the start of next Display Pipe A “Horizontal Blank” event occurs. This event is defined as the start of the next Display A Horizontal blank period. Note that this can cause a wait for up to a line. See Horizontal Blank Event in the Device Programming Interface chapter of *MI Functions*.
15	**Display Plane B Asyncronous Performance Flip Pending Wait Enable** Project: All Format: Enable This field enables a wait for the duration of a Display Plane B “Flip Pending” condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of *MI Functions*.
14	**Display Plane B Asyncronous Flip Pending Wait Enable** Project: All Format: Enable This field enables a wait for the duration of a Display Plane B “Flip Pending” condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of *MI Functions*.

SYNC_FLIP_STATUS – Wait for event and Display flip flags Register	
13	**Display Plane B Syncronous Flip Pending Wait Enable** Project: All Format: Enable This field enables a wait for the duration of a Display Plane B “Flip Pending” condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of *MI Functions*.
12	**Display Sprite B Syncronous Flip Pending Wait Enable** Project: All Format: Enable This field enables a wait for the duration of a Display Sprite B “Flip Pending” condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of *MI Functions*.
11	**Reserved** Project: All Format: MBZ
10	**Display Pipe B Scan Line Wait Enable** Project: All Format: Enable This field enables a wait while a Display Pipe B “Scan Line” condition exists. This condition is defined as the the start of the scan line specified in the Pipe B Display Scan Line Count Range Compare Register. See Scan Line Event in the Device Programming Interface chapter of *MI Functions*.
9	**Display Pipe B Vertical Blank Wait Enable** Project: All Format: U32 This field enables a wait until the next Display Pipe B “Vertical Blank” event occurs. This event is defined as the start of the next Display Pipe B vertical blank period. Note that this can cause a wait for up to an entire refresh period. See Vertical Blank Event (See *Programming Interface*).
8	**Display Pipe B H Blank Wait Enable** Project: All Format: Enable This field enables a wait until the start of next Display Pipe B “Horizontal Blank” event occurs. This event is defined as the start of the next Display B Horizontal blank period. Note that this can cause a wait for up to a line. See Horizontal Blank Event in the Device Programming Interface chapter of *MI Functions*.
7:5	**Reserved** Project: All Format: MBZ

SYNC_FLIP_STATUS – Wait for event and Display flip flags Register

4:0

Condition Code Wait Select

Project: All

This field enables a wait for the duration that the corresponding condition code is active. These enable select one of 15 condition codes in the EXCC register, that cause the parser to wait until that condition-code in the EXCC is cleared.

Value	Name	Description	Project
0h	Not Enabled	Condition Code Wait not enabled	All
1h-5h	Enabled	Condition Code select enabled; selects one of 5 codes, 0 – 4	All
6h-15h	Reserved		All

Programming Notes
Note that not all condition codes are implemented. The parser operation is UNDEFINED if an unimplemented condition code is selected by this field. The description of the EXCC register (*Memory Interface Registers*) lists the codes that are implemented.

1.1.10 Pipelines Statistics Counter Registers

These registers keep continuous count of statistics regarding the 3D pipeline. They are saved and restored with context but should not be changed by software except to reset them to 0 at context creation time. These registers may be read at any time; however, to obtain a meaningful result, a pipeline flush just prior to reading the registers is necessary in order to synchronize the counts with the primitive stream.

1.1.10.1 IA_VERTICES_COUNT — Reported Vertices Counter

IA_VERTICES_COUNT	
Register Type:	MMIO_CS
Address Offset:	2310h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	64
Trusted Type:	1
This register stores the count of vertices processed by VF. This register is part of the context save and restore.	

Bit	Description
63:0	**IA Vertices Count Report** Total number of vertices fetched by the VF stage. This count is updated for every input vertex as long as **Statistics Enable** is set in VF_STATE (see the Vertex Fetch Chapter in the *3D* Volume.)

1.1.10.2 IA_PRIMITIVES_COUNT — Reported Vertex Fetch Output Primitives Counter

IA_PRIMITIVES_COUNT	
Register Type:	MMIO_CS
Address Offset:	2318h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	64
Trusted Type:	1
This register stores the count of primitives generated by VF. This register is part of the context save and restore.	

Bit	Description
63:0	**IA Primitives Count Report** Total number of primitives output by the Vertex Fetch (IA) stage. This count is updated for every primitive *output* by the VF stage, as long as **Statistics Enable** is set in VF_STATE (see the Vertex Fetch Chapter in the *3D* Volume.)

1.1.10.3 VS_INVOCATION_COUNT— Reported Vertex Shader Invocation Counter

VS_INVOCATION_COUNT	
Register Type:	MMIO_CS
Address Offset:	2320h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	64
Trusted Type:	1
This register stores the value of the vertex count shaded by VS. This register is part of the context save and restore	

Bit	Description
63:0	**VS Invocation Count Report** Number of vertex shader threads invoked by the VS stage. Updated only when **Statistics Enable** is set in VS_STATE (see the Vertex Shader Chapter in the *3D* Volume.)

1.1.10.4 GS_INVOCATION_COUNT — Reported Geometry Shader Thread Invocation Counter

GS_INVOCATION_COUNT	
Register Type:	MMIO_CS
Address Offset:	2328h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	64
Trusted Type:	1

This register stores the number of invoked geometry shader threads. This register is part of the context save and restore.

Bit	Description
63:0	**GS Invocation Count** Number of geometry shader threads invoked by the GS stage. Updated only when **Statistics Enable** is set in GS_STATE (see the Geometry Shader Chapter in the *3D* Volume.)

1.1.10.5 GS_PRIMITIVES_COUNT — Reported Geometry Shader Output Primitives Counter

GS_PRIMITIVES_COUNT	
Register Type:	MMIO_CS
Address Offset:	2330h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	64
Trusted Type:	1

This register reflects the total number of primitives that have been output by the Geometry Shader stage. This register is part of the context save and restore.

Bit	Description
63:0	**GS Primitives Count** Total number of primitives output by the geometry stage. Updated only when **Statistics Enable** is set in GS_STATE (see the Geometry Shader Chapter in the *3D* Volume.)

1.1.10.6 CL_INVOCATION_COUNT— Reported Clipper Thread Invocation Counter

CL_INVOCATION_COUNT	
Register Type:	MMIO_CS
Address Offset:	2338h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	64
Trusted Type:	1

This register stores the count of objects entering the Clipper stage. This register is part of the context save and restore.

Bit	Description
63:0	**CL Invocation Count Report** Number of objects entering the clipper stage. Updated only when **Statistics Enable** is set in CLIP_STATE (see the Clipper Chapter in the *3D* Volume.)

1.1.10.7 CL_PRIMITIVES_COUNT— Reported Clipper Output Primitives Counter

CL_PRIMITIVES_COUNT	
Register Type:	MMIO_CS
Address Offset:	2340h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	64
Trusted Type:	1
This register reflects the total number of primitives that have been output by the clipper. This register is part of the context save and restore.	
Bit	**Description**
63:0	**Clipped Primitives Output Count** Total number of primitives output by the clipper stage. This count is updated for every primitive *output* by the clipper stage, as long as **Statistics Enable** is set in SF_STATE (see the Clipper and SF Chapters in the *3D* Volume.)

1.1.10.8 PS_INVOCATION_COUNT— Reported Pixels Shaded Counter

PS_INVOCATION_COUNT	
Register Type:	MMIO_CS
Address Offset:	2348h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	64
Trusted Type:	1
This register stores the value of the count of fragments that get shaded. This register is part of the context save and restore.	
Bit	**Description**
63:0	**PS Invocation Count** *Reflects a count of the total number of fragments that are dispatched to pixel shader invocations while **Statistics Enable** is set in the Windower. See the Windower chapter of the 3D volume for details. This count will generally be much greater than the actual count of PS threads since a single thread may process up to 32 pixels.*

1.1.10.9 PS_DEPTH_COUNT — Reported Pixels Passing Depth Test counter

PS_DEPTH_COUNT	
Register Type:	MMIO_CS
Address Offset:	2350h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	64
Trusted Type:	1

This register stores the value of the count of pixels that have passed the depth test. This register is part of the context save and restore. Note that the value of this register can be obtained in a pipeline-synchronous fashion without a pipeline flush by using the 3DCONTROL command. See 3D Overview in the 3D volume.

Bit	Description
63:0	**Depth Count** This register reflects the total number of pixels that have passed the depth test (i.e., will be visible). All pixels are counted when **Statistics Enable** is set in the Windower State. See the Windower chapter of the *3D* volume for details. Pixels that pass the depth test but fail the stencil test will *not* be counted.

1.1.10.10 TIMESTAMP — Reported Timestamp Count

TIMESTAMP — Reported Timestamp Count	
Register Type:	MMIO_CS
Address Offset:	2358h
Project:	All
Default Value:	0000 0000 0000 0000h
Access:	RO. This register is *not* set by the context restore.
Size (in bits):	64

This register provides an elapsed real-time value that can be used as a timestamp for GPU events over short periods of time. Note that the value of this register can be obtained in a 3D pipeline-synchronous fashion without a pipeline flush by using the PIPE_CONTROL command. See 3D Geometry Pipeline in the "3D and Media" volume.

This register (effectively) counts at a constant frequency by adjusting the increment amount according to the actual reference clock frequency. SW therefore does not need to know the reference clock frequency.

This register is not reset by a graphics reset. It will maintain its value unless a full chipset reset is performed.

Bit	Description
63:36	**Reserved** Project: All Format: MBZ
35:0	**Timestamp Value** Project: All Format: U32 This register toggles every 80 ns of time.

1.1.10.11 SO_NUM_PRIMS_WRITTEN— Reported Stream Output Num Primitives Written Counter

SO_NUM_PRIMS_WRITTEN— Reported Stream Output Num Primitives Written Counter	
Register Type:	MMIO_CS
Address Offset:	2288h
Project:	All
Default Value:	0000 0000 0000 0000h
Access:	R/W
Size (in bits):	64

This register is used to (indirectly) count the number of primitives which GS threads have successfully written to Streamed Vertex Output buffers. This register is part of the context save and restore.

[Errata] This register gets reset when write happens to register 2380h

Bit	Description
63:0	**Num Prims Written Count** Project: All Format: U64 This count is incremented (by one) every time a GS thread outputs a DataPort Streamed Vertex Buffer Write message with the **Increment Num Prims Written** bit set in the message header (see the *Geometry Shader* and *Data Port* chapters in the *3D* Volume.)

1.1.10.12 SO_PRIM_STORAGE_NEEDED — Reported Stream Output Primitive Storage Needed Counter

SO_PRIM_STORAGE_NEEDED — Reported Stream Output Primitive Storage Needed Counter	
Register Type:	MMIO_CS
Address Offset:	2280h
Project:	All
Default Value:	0000 0000 0000 0000h
Access:	RO. This register is set by the context restore.
Size (in bits):	64

This register is used to (indirectly) count the number of primitives which GS threads would have written to Streamed Vertex Output buffers if all buffers had been large enough to accommodate the writes . This register is part of the context save and restore.
[Errata] This register gets reset when write happens to register 2388h

Bit	Description
63:0	**Prim Storage Needed Count** Project: All Format: U64 This count is incremented (by one) every time a GS thread outputs a DataPort Streamed Vertex Buffer Write message with the **Increment Prim Storage Needed** bit set in the message header (see the *Geometry Shader* and *Data Port* chapters in the *3D* Volume.)

1.1.11 Performance Statistics Registers

[DevSNB] When an over flow condition occurs and the buffers need to be reset, or when software wants to change the OABUFFER to point to a new area in memory, Programming of the performance ring must follow the sequence below.

- Clear OA enable bit by writing 0x2360[0] = 0
- Write OASTATUS2
- Write OABUFFER
- Write OASTATUS1
- Set OA enable bit by writing 0x2360[0] = 1

1.1.11.1 OACONTROL – Observation Architecture Control

OACONTROL – Observation Architecture Control

Register Type: MMIO
Address Offset: [DevSNB] 2360h
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 32

This register is used to program the OA unit.

Bit	Description
31:12	**Select Context ID** Project: All Specifies the context ID of the one context that affects the performance counters. All other contexts are ignored.
11:6	**Timer Period** Project: All Format: Select Specifies the period of the timer strobe as a function of the minimum TIME_STAMP resolution. The period is determined by selecting a specified bit from the TIME_STAMP register as follows: $\text{StrobePeriod} = \text{MinimumTimeStampPeriod} * 2^{\text{TimerPeriod}}$ The exponent is defined by this field. Note: The TIME_STAMP is not reset at start time so the phase of the strobe is not synchronized with the enable of the OA unit. This could result in approximately a full StrobePeriod elapsing prior to the first trigger. Usage for this mechanism should be time based periodic triggering, typically.
5	**Timer Enable** Project: All Default Value: 0h Disabled Format: Enable This field enables the timer logic to output a periodic strobe, as defined by the Timer Period. When disabled the timer output is not asserted.

Value	Name	Description	Project
0h	Disable	Counter does not get written out on regular interval	All
1h	Enable	Counter gets written out on regular intervals, defined by the **Timer Period**	All

4:2	**Reserved**

OACONTROL – Observation Architecture Control

1	**Specific Context Enable** Project: [DevSNB] Default Value: 0h All contexts considered Mask: MMIO(0x2000)#16 Format: U32 FormatDesc Enables counters to work on a context specific workload. The context is given by bits 31:12. OA unit level clock gating must be ENABLED when using specific ContextID feature.

Value	Name	Description	Project
0h	Disable	All contexts are considered	All
1h	Enable	Only the contexts with the **Select Context ID** are considered	All

0	**Performance Counter Enable** Project: All Format: Enable Global performance counter enable. If clear, no counting will occur. MI_REPORT_PERF_COUNT is undefined when clear. **[DevSNB]** When this bit is set, in order to have cohenret counts, RC6 power state must be disabled. This can be achieved by programming MMIO registers as 0xA094=0x0 and 0xA090[31]=1. **[DevSNB]** EU clock gating must be disabled when this bit is set.

1.1.11.2 OASTATUS1 – Observation Architecture Status Register

OASTATUS1—Observation Architecture Status Register

Register Type: MMIO
Address Offset: [DevSNB] 2364h
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 32

This register is used to program the OA unit.

Bit	Description
31:6	**Tail Ppointer** Project: All Virtual address of the internal trigger based buffer and it is updated for every 64B cacheline write to memory when reporting via internal trigger. This pointer will not be updated for MI_REPORT_PERF_COUNT command based writes. When OA is enabled, this address must be programmed by SW to the base address of the internal trigger base mechanism. [DevSNB+]: SW must ensure that Tail pointer and the Head Pointer (in OASTATUS2) do not have different values while programming.
5:3	**Inter Trigger Report Buffer Size** Project: All Default Value: 0h All context considered This field indicates the size of buffer for internal trigger mechanism. This field is programmed in terms of multiple of 128KB.

Value	Description	Project
0b	16KB	GEN6
1b	32KB	GEN6
2	48KB	GEN6
3	64KB	GEN6
4	80KB	GEN6
5	96KB	GEN6
6	112KB	GEN6
7	128KB	GEN6

OASTATUS1—Observation Architecture Status Register	
2	**Counter OverFlow Error** Project: All Format: Select This bit is set if any of the counters overflows. This bit can be reset by SW in B0. [DevSNB] Erratum: This bit must be cleared after the ring is enabled and before OA is enabled.
1	**Buffer Overflow** Project: All Default Value: 0h This bit is set when the Tail-pointer - Head pointer > max internal trigger buffer size
0	**Report Lost Error** Project: All Format: Enable This bit is set if the Report Logic is requested to write out the counter values before the previous report request was completed. The report request is ignored and the counter continue to count. This bit can be reset by SW in B0.

1.1.11.3 OASTATUS2 – Observation Architecture Status Register

OASTATUS2—Observation Architecture Status Register	
Register Type:	MMIO
Address Offset:	[DevSNB] 2368h
Project:	All
Default Value:	00000000h
Access:	RW
Size (in bits):	32
This register is used to program the OA unit.	
Bit	**Description**
31:6	**Head Pointer** Project: All Virtual address of the internal trigger based buffer that is updated by software after consuming from the report buffer. This pointer must be updated by SW for internal trigger base buffer only.
4:1	**Reserved** Project: All Format: MBZ

OASTATUS2—Observation Architecture Status Register	
0	**Memory select PPGTT/GGTT access** Project: All Format: U32 0 – PPGTT 1 – GGTT

1.1.11.4 OABUFFER – Observation Architecture Buffer

OABUFFER—Observation Architecture Status Register	
Register Type: MMIO **Address Offset:** [DevSNB] 23B0h **Project:** All **Default Value:** 00000000h **Access:** {DevSNB] Write Only **Size (in bits):** 32	
This register is used to program the OA unit. [DevSNB] This MMIO must be set *before* the **OASTATUS1** register and set *after* the OASTATUS2 register. This is to enable proper functionality of the overflow bit. [DevSNB] Report Buffer Offset Must be 512KB aligned.	
Bit	**Description**
31:6	**Report Buffer Offset** Project: All This field specifies 64B aligned GFX MEM address where the chap counter values are reported.
5	**Reserved** Project: All Format: MBZ
4	**Reserved**
3	**Reserved**

OABUFFER—Observation Architecture Status Register	
2	**OA Report Trigger Select** Project: All Format: 1 - Level Report trigger 1-2 - Edge Report trigger.
1	**Reserved**
0	**Reserved**

1.1.11.5 OASTARTTRIG1 – Observation Architecture Start Trigger

OASTARTTRIG1—Observation Architecture Buffer	
Register Type:	MMIO
Address Offset:	[DevSNB] 238Ch
Project:	All
Default Value:	00000000h
Access:	RW
Size (in bits):	32
This register is used to program the OA unit. **OASTARTTRIG5-8 will be used to start Boolean counters 4 to 7.** **OASTARTTRIG1-4 will be used to start Boolean counters 0 to 3.** GEN6 report trigger behavior can be derived by programming these two sets of OA START registers with the same value.	
Bit	**Description**
31:16	**Reserved** Project: All Format: MBZ
15:0	**Threshold Value** Project: All Format: U16 Threshold value for the compare logic within the trigger logic

1.1.11.6 OASTARTTRIG2 – Observation Architecture Start Trigger

OASTARTTRIG2— Observation Architecture Start Trigger	
Register Type:	MMIO
Address Offset:	[DevSNB] 2388h
Project:	All
Default Value:	00000000h
Access:	RW
Size (in bits):	32

This register is used to program the OA unit.

OASTARTTRIG5-8 will be used to start Boolean counters 4 to 7.

OASTARTTRIG1-4 will be used to start Boolean counters 0 to 3.

GEN6 report trigger behavior can be derived by programming these two sets of OA START registers with the same value.

Bit	Description
31:24	**Reserved**
23	**Threshold Enable** Enable the threshold compare logic within the trigger logic.
22	**vert D Enable 0** Invert the specified signal at the D stage of the trigger logic
21	**Invert C Enable 1** Invert the specified signal at the C stage of the trigger logic.
20	**Invert C Enable 0** Invert the specified signal at the C stage of the trigger logic.
19	**Invert B Enable 3** Invert the specified signal at the B stage of the trigger logic.
18	**Invert B Enable 2** Invert the specified signal at the B stage of the trigger logic
17	**Invert B Enable 1** Invert the specified signal at the B stage of the trigger logic
16	**Invert B Enable 0** Invert the specified signal at the B stage of the trigger logic

OASTARTTRIG2—Observation Architecture Start Trigger	
15	**Invert A Enable 15** Invert the specified signal at the A stage of the trigger logic.
14	**Invert A Enable 14** Invert the specified signal at the A stage of the trigger logic.
13	**Invert A Enable 13** Invert the specified signal at the A stage of the trigger logic.
12	**Invert A Enable 12** Invert the specified signal at the A stage of the trigger logic.
11	**Invert A Enable 11** Invert the specified signal at the A stage of the trigger logic.
10	**Invert A Enable 10** Invert the specified signal at the A stage of the trigger logic.
9	**Invert A Enable 9** Invert the specified signal at the A stage of the trigger logic.
8	**Invert A Enable 8** Invert the specified signal at the A stage of the trigger logic
7	**Invert A Enable 7** Invert the specified signal at the A stage of the trigger logic.
6	**Invert A Enable 6** Invert the specified signal at the A stage of the trigger logic.
5	**Invert A Enable 5** Invert the specified signal at the A stage of the trigger logic.
4	**Invert A Enable 4** Invert the specified signal at the A stage of the trigger logic.
3	**Invert A Enable 3** Invert the specified signal at the A stage of the trigger logic.

OASTARTTRIG2—Observation Architecture Start Trigger	
2	**Invert A Enable 2** Invert the specified signal at the A stage of the trigger logic.
1	**Invert A Enable 1** Invert the specified signal at the A stage of the trigger logic.
0	**Invert A Enable 0** Invert the specified signal at the A stage of the trigger logic.

1.1.11.7 OAREPORTTRIG1 – Observation Architecture Report Trigger

OAREPORTTRIG1—Observation Architecture Report Trigger

Register Type: MMIO
Address Offset: [DevSNB] 237Ch
Project: All
Default Value: 00000000h
Access: RW
Size (in bits): 32

This register is used to program the OA unit.

Bit	Description
31:16	**Occurrence vs. Duration Select** Project: All Format: Occurrence[16] 1 bit per NOA counter total 16 bits
15:0	**Threshold Value** Project: All Format: U16 Threshold value for the compare logic within the trigger logic

Value	Name	Description	Project
0h	Duration		All
1h	Occurence		All

1.1.11.8 OAREPORTTRIG2 – Observation Architecture Report Trigger

OAREPORTTRIG2—Observation Architecture Report Trigger	
Register Type:	MMIO
Address Offset:	[DevSNB] 2378h
Project:	All
Default Value:	00000000h
Access:	RW
Size (in bits):	32

This register is used to program the OA unit.

Bit	Description
31	**Report Trigger enable** **Enable the report trigger for threshold triggers**.
30:24	**Reserved** Project: All Format: MBZ
23	**Threshold Enable** Enable the threshold compare logic within the trigger logic.
22	**Invert D Enable 0** Invert the specified signal at the D stage of the trigger logic.
21	**Invert C Enable 1** Invert the specified signal at the C stage of the trigger logic.
20	**Invert C Enable 0** Invert the specified signal at the C stage of the trigger logic.
19	**Invert B Enable 3** Invert the specified signal at the B stage of the trigger logic.
18	**Invert B Enable 2** Invert the specified signal at the B stage of the trigger logic.
17	**Invert B Enable 1** Invert the specified signal at the B stage of the trigger logic.
16	**Invert B Enable 0** Invert the specified signal at the B stage of the trigger logic.
15	**Invert A Enable 15** Invert the specified signal at the A stage of the trigger logic.

OAREPORTTRIG2—Observation Architecture Report Trigger	
14	**Invert A Enable 14** Invert the specified signal at the A stage of the trigger logic
13	**Invert A Enable 13** Invert the specified signal at the A stage of the trigger logic
12	**Invert A Enable 12** Invert the specified signal at the A stage of the trigger logic
11	**Invert A Enable 11** Invert the specified signal at the A stage of the trigger logic
10	**Invert A Enable 10** Invert the specified signal at the A stage of the trigger logic
9	**Invert A Enable 9** Invert the specified signal at the A stage of the trigger logic
8	**Invert A Enable 8** Invert the specified signal at the A stage of the trigger logic
7	**Invert A Enable 7** Invert the specified signal at the A stage of the trigger logic
6	**Invert A Enable 6** Invert the specified signal at the A stage of the trigger logic
5	**Invert A Enable 5** Invert the specified signal at the A stage of the trigger logic
4	**Invert A Enable 4** Invert the specified signal at the A stage of the trigger logic
3	**Invert A Enable 3** Invert the specified signal at the A stage of the trigger logic
2	**Invert A Enable 2** Invert the specified signal at the A stage of the trigger logic
1	**Invert A Enable 1** Invert the specified signal at the A stage of the trigger logic
0	**Invert A Enable 0** Invert the specified signal at the A stage of the trigger logic

1.1.11.9 CEC0-0 – Customizable Event Creation

CEC0-0— Customizable Event Creation

Register Type:	MMIO
Address Offset:	[DevSNB] 2390h
Project:	All
Default Value:	00000000h
Access:	Write Only
Size (in bits):	32

This register is used to program the OA unit.

Bit	Description
31:21	**Reserved** Project: All Format: MBZ
20:19	**Source Select** Project: All Format: U2 Selects Event for the Boolean logic. Selects the 16 bunch of events from the, Boolean Events and Previous Events.
18:3	**Compare Value** Project: All Format: U16 This field is loaded to compare against the 8 NOA signals that are fed into this block. The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the NOA event is asserted. This in turn can be counted by any of the CHAP counters.

Value	Name	Description	Project
00b	Reserved		All
01b	Prev Events	Selects the Previous events	All
10b	Boolean Events	Selects the Boolean Events	All
11b	Reserved		All

CEC0-0— Customizable Event Creation	
2:0	**Compare Function** Project: All Format: U16 This field is loaded to compare against the 8 NOA signals that are fed into this block. The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the NOA event is asserted. This in turn can be counted by any of the CHAP counters.

Value	Name	Description	Project
000b	Any Are Equal	Compare and assert if any are equal (Can be used as OR function)	All
001b	Greater Than	Compare and output signal if greater than	All
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)	All
011b	Greater Than or Equal	Compare and assert output if greater than or equal	All
100b	Less Than	Compare and assert output if less than	All
101b	Not Equal	Compare and assert output if not equal	All
110b	Less Than or Equal	Compare and assert output if less than or equal	All
111b	Reserved		All

1.1.11.10 CEC0-1 – Customizable Event Creation

CEC0-1— Customizable Event Creation	
Register Type:	MMIO
Address Offset:	[DevSNB] 2394h
Project:	All
Default Value:	00000000h
Access:	Write Only
Size (in bits):	32
This register is used to program the OA unit.	

Bit	Description
31:16	**Reserved**
15:0	**Mask** Project: All Format: U32 These 8 bits are used to mask off entries from the comparison. For each bit: 0: This bit is considered in event calculations. 1: This bit is ignored in event calculations.

1.1.11.11 CEC1-0 – Customizable Event Creation

CEC1-0— Customizable Event Creation

Register Type: MMIO
Address Offset: [DevSNB] 2398h

Project: All
Default Value: 00000000h
Access: Write Only
Size (in bits): 32

This register is used to program the OA unit.

Bit	Description
31:21	**Reserved** Project: All Format: MBZ
20:19	**Source select** Project: All Format: U2 Selects Event for the Boolean logic. Selects the 16 bunch of events from the Booleanents and Previous Events.
18:3	**Reserved** Project: All Format:
2:0	**Compare Function** Project: All Format: U3

Value	Name	Description	Project
00b	Reserved		All
01b	Prev Events	Selects the Previous events	All
10b	Boolean Events	Selects the Boolean Events	All
11b	Reserved		All

Value	Name	Description	Project
000b	Any Are Equal	Compare and assert if any are equal (Can be used as OR function)	All
001b	Greater Than	Compare and output signal if greater than	All
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)	All
011b	Greater Than or Equal	Compare and assert output if greater than or equal	All
100b	Less Than	Compare and assert output if less than	All
101b	Not Equal	Compare and assert output if not equal	All
110b	Less Than or Equal	Compare and assert output if less than or equal	All
111b	Reserved		All

1.1.11.12 CEC1-1 – Customizable Event Creation

CEC1-1— Customizable Event Creation	
Register Type:	MMIO
Address Offset:	[DevSNB] 239Ch
Project:	All
Default Value:	00000000h
Access:	RW
Size (in bits):	32
This register is used to program the OA unit.	

Bit	Description
31:16	**Considerations** Project: All Format: U32 0: The bit is considered in event calculations. 1: The bit is delayed by 1 clock before considering it in event calculations. This is particularly useful for doing state machine arc coverage
15:0	**Mask** Project: All Format: U32 These 8 bits are used to mask off entries from the comparison. For each bit: 0: This bit is considered in event calculations. 1: This bit is ignored in event calculations.

1.1.11.13 CEC2-0 – Customizable Event Creation

CEC2-0— Customizable Event Creation

Register Type: MMIO
Address Offset: [DevSNB] 23A0h

Project: All
Default Value: 00000000h
Access: Write Only
Size (in bits): 32

This register is used to program the OA unit.

Bit	Description
31:21	**Reserved** Project: All Format: MBZ
20:19	**Source select** Project: All Format: U2 Selects Event for the Boolean logic. Selects the 16 bunch of events from the Boolean Events and Previous Events. See section 5 for more details
18:3	**Compare Value** Project: All Format: U16 The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the NOA event is asserted. This in turn can be counted by any of the CHAP counters.
2:0	**Compare Function** Project: All Format: U3

Value	Name	Description	Project
00b	Reserved		All
01b	Prev Events	Selects the Previous events	All
10b	Boolean Events	Selects the Boolean Events	All
11b	Reserved		All

Value	Name	Description	Project
000b	Any Are Equal	Compare and assert if any are equal (Can be used as OR function)	All
001b	Greater Than	Compare and output signal if greater than	All
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)	All
011b	Greater Than or Equal	Compare and assert output if greater than or equal	All
100b	Less Than	Compare and assert output if less than	All
101b	Not Equal	Compare and assert output if not equal	All
110b	Less Than or Equal	Compare and assert output if less than or equal	All
111b	Reserved		All

1.1.11.14 CEC2-1 – Customizable Event Creation

CEC2-1— Customizable Event Creation					
Register Type:	MMIO				
Address Offset:	[DevSNB] 23A4h				
Project:	All				
Default Value:	00000000h				
Access:	Write Only				
Size (in bits):	32				
This register is used to program the OA unit.					
Bit	**Description**				
31:16	**Considerations**	Project:	All	Format:	U32
	0: The bit is considered in event calculations. 1: The bit is delayed by 1 clock before considering it in event calculations. This is particularly useful for doing state machine arc coverage.				
15:0	**Mask**	Project:	All	Format:	U32
	These 8 bits are used to mask off entries from the comparison. For each bit: 0: This bit is considered in event calculations. 1: This bit is ignored in event calculations.				

1.1.11.15 CEC3-0 – Customizable Event Creation

CEC3-0— Customizable Event Creation

Register Type: MMIO
Address Offset: [DevSNB] 23A8h

Project: All
Default Value: 00000000h
Access: RW
Size (in bits): 32

This register is used to program the OA unit.

Bit	Description
31:21	**Reserved** Project: All Format: MBZ
20:19	**Source select** Project: [DevSNB] Format: U2 Selects Event for the Boolean logic. Selects the 16 bunch of events from the Boolean Events and Previous Events.

Value	Name	Description	Project
00b	Reserved		All
01b	Prev Events	Selects the Previous events	All
10b	Boolean Events	Selects the Boolean Events	All
11b	Reserved		All

Bit	Description
18:3	**Compare Value** Project: All Format: U16 This field is loaded to compare against the 8 signals that are fed into this block. The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the event is asserted. This in turn can be counted by any of the CHAP counters.
2:0	**Compare Function** Project: All Format: U3

Value	Name	Description	Project
000b	Any Are Equal	Compare and assert if any are equal (Can be used as OR function)	All
001b	Greater Than	Compare and output signal if greater than	All
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)	All
011b	Greater Than or Equal	Compare and assert output if greater than or equal	All
100b	Less Than	Compare and assert output if less than	All
101b	Not Equal	Compare and assert output if not equal	All
110b	Less Than or Equal	Compare and assert output if less than or equal	All
111b	Reserved		All

1.1.11.16 CEC3-1 – Customizable Event Creation

CEC3-1— Customizable Event Creation	
Register Type:	MMIO
Address Offset:	{DevSNB] 23ACh
Project:	All
Default Value:	00000000h
Access:	Write Only
Size (in bits):	32
This register is used to program the OA unit.	
Bit	**Description**
31:16	**Considerations** Project: All Format: U32 0: The bit is considered in event calculations. 1: The bit is delayed by 1 clock before considering it in event calculations. This is particularly useful for doing state machine arc coverage.
15:0	**Mask** Project: All Format: U32 These 8 bits are used to mask off entries from the comparison. For each bit: 0: This bit is considered in event calculations. 1: This bit is ignored in event calculations.

1.2 Memory Interface Commands for Rendering Engine

1.2.1 Introduction

This chapter describes the formats of the "Memory Interface" commands, including brief descriptions of their use. The functions performed by these commands are discussed fully in the *Memory Interface Functions* Device Programming Environment chapter.

This chapter describes MI Commands for the original graphics processing engine. The term "for Rendering Engine" in the title has been added to differentiate this chapter from a similar one describing the MI commands for the Media Decode Engine.

The commands detailed in this chapter are used across products within the Gen4+ family. However, slight changes may be present in some commands (i.e., for features added or removed), or some commands may be removed entirely. Refer to the *Preface* chapter for product specific summary.

1.2.2 Software Synchronization Commands

To support mid-triangle interruption, certain commands need to be placed in a temporary location in hardware until primitive commands are complete. This introduces out-of-order command execution. Below show the commands that are affected. Note that the INSTPM register has a bit that is used to force in-order execution. If set, however, mid-triangle modes like PSMI cannot be enabled.

Command	Qualifications
MI_NOOP	When writing to the NOOPID register
MI_USER_INTERRUPT	Always
MI_PROBE	Writing out new value after check
MI_UNPROBE	Always
MI_SEMAPHORE_MBOX	Memory write
MI_STORE_DATA_IMM	Always
MI_STORE_DATA_INDEX	Always
MI_LOAD_REGISTER_IMM	Always
MI_UPDATE_GTT	Always
MI_STORE_REGISTER_MEM	Register read is done in-order, register write done out-of-order

1.2.3 MI_ARB_CHECK

MI_ARB_CHECK

Project:	All	**Length Bias:**	1
Engine:	**Render**		

The MI_ARB_CHECK instruction is used to check the ring buffer double buffered head pointer (register UHPTR). This instruction can be used to pre-empt the current execution of the ring buffer. Note that the valid bit in the updated head pointer register needs to be set for the command streamer to be pre-empted.

Programming Notes:

- The current head pointer is loaded with the updated head pointer register independent of the location of the updated head
- If the current head pointer and the updated head pointer register are equal, hardware will automatically reset the valid bit corresponding to the UHPTR
- For GEN6 this instruction can be placed only in a ring buffer, never in a batch buffer.
- For pre-emption, the wrap count in the ring buffer head register is no longer maintained by hardware. The hardware updates the wrap count to the value in the UHPTR register.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 05h MI_ARB_CHECK Format: OpCode
	22:0	**Reserved** Project: All Format: MBZ

1.2.4 MI_ARB_ON_OFF

MI_ARB_ON_OFF			
Project:	All	**Length Bias:**	1
Engine:	**Render**		

The MI_ARB_ON_OFF instruction is used to disable/enable context switching. Note that context switching will remain disabled until re-enabled through use of this command. This command will also prevent a switch in the case of waiting on events, running out of commands or a surface probe fault. These will effectively hang the device if allowed to occur while arbitration is off (context switching is disabled.)

This command should always be used as an off-on pair with the sequence of instructions to be protected from context switch between MI_ARB_OFF and MI_ARB_ON. Software must use this arbitration control with caution since it has the potential to increase the response time of the Render Engine to pre-emption requests.

This is a privileged command; it will not be effective (will be converted to a no-op) if executed from within a non-secure batch buffer. This command can only be issued when **Per-Process Virtual Address Space** is set; if the bit is set it will be converted to NOOP.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 08h MI_ARB_ON_OFF Format: OpCode
	22:1	**Reserved** Project: All Format: MBZ
	0	**Arbitration Enable** Format: Enable This field enables or disables context switches due to pre-emption.

1.2.5 MI_BATCH_BUFFER_END

MI_BATCH_BUFFER_END			
Project:	All	**Length Bias:**	1
Engine:	**Render**		

The MI_BATCH_BUFFER_END command is used to terminate the execution of commands stored in a *batch buffer* initiated using a MI_BATCH_BUFFER_START command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Ah MI_ BATCH_BUFFER_END Format: OpCode
	22:0	**Reserved** Project: All Format: MBZ

1.2.6 MI_CONDITIONAL_BATCH_BUFFER_END

MI_CONDITIONAL_BATCH_BUFFER_END			
Project:	All	**Length Bias:**	2
Engine:	**Render**		

The MI_BATCH_BUFFER_END command is used to conditionally terminate the execution of commands stored in a *batch buffer* initiated using a MI_BATCH_BUFFER_START command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 36h MI_CONDITIONAL_BATCH_BUFFER_END Format: OpCode
	22	**Use Global GTT** Project: All Default Value: 0h DefaultVaueDesc Format: U1 FormatDesc If set, this command will use the global GTT to translate the **Compare Address** and this command must be executing from a privileged (secure) batch buffer. If clear, the PPGTT will be used to translate the **Compare Address**.

MI_CONDITIONAL_BATCH_BUFFER_END		
	21	**Compare Semaphore** Project: All Default Value: 0h DefaultVaueDesc Format: U1 FormatDesc If set, the value from the **Compare Data Dword** is compared to the value from the **Compare Address** in memory. If the value at **Compare Address is greater than the Compare Data Dword**, execution of current command buffer should continue. If clear, no comparison takes place.
	20	**Reserved**
	19:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:0	**Compare Data Dword** Data dword to compare memory. The Data dword is supplied by software to control execution of the command buffer. If the compare is enabled and the data at **Compare Address** is greater than this dword, the execution of the command buffer should continue.
2	31:3	**Compare Address** Qword address to fetch Data Dword(DW0) from memory. HW will compare the Data Dword(DW0) with **Compare** Data Dword
	2:0	**Reserved** Project: All Format: MBZ

1.2.7 MI_BATCH_BUFFER_START

MI_BATCH_BUFFER_START

Project:	All	**Length Bias:**	2
Engine:	**Render**		

The MI_BATCH_BUFFER_START command is used to initiate the execution of commands stored in a *batch buffer.* For restrictions on the location of batch buffers, see Batch Buffers in the Device Programming Interface chapter of *MI Functions*.

The batch buffer can be specified as secure or non-secure, determining the operations considered valid when initiated from within the buffer and any attached (chained) batch buffers. See Batch Buffer Protection in the Device Programming Interface chapter of *MI Functions*.

Programming Notes:

- Batch buffers referenced with physical addresses must not extend beyond the end of the starting physical page (can't span physical pages). However, a batch buffer initiated using a physical address can chain to another buffer in another physical page.
- A batch buffer initiated with this command must end either with a MI_BATCH_BUFFER_END command or by chaining to another batch buffer with an MI_BATCH_BUFFER_START command.
- For virtual batch buffers, it is essential that the address location beyond the current page be populated inside the GTT. HW performs over-fetch of the command addresses and any over-fetch requires a valid TLB entry. A single extra page beyond the batch buffer is sufficient.
- Prior to sending batch buffer start command with clear command buffer enable set, software has to ensure pipe is flushed explicitly by sending MI_FLUSH.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 31h MI_BATCH_BUFFER_START Format: OpCode
	22	**Reserved** Format: MBZ
	22:17	**Reserved** Project: All Format: MBZ
	16	Reserved Project: Format: .
	15	**Reserved** Project: Format:

MI_BATCH_BUFFER_START

	14:13	**Reserved** Project: All Format: MBZ
	12	**Reserved** Project: All Format:
	11	**Clear Command Buffer Enable** Project: All Format: U1 The following batch buffer is to be executed from the Write Once protected memory area. The address of the batch buffer is an offset into the WOPCM area. This batch buffer needs to be pre-ceded by a MI_FLUSH command or PIPE_CONTROL with CS Stall set.
	10	**Reserved** Format: MBZ
	9	**Reserved** Project: All Format: MBZ
	8	**Buffer Security and Address Space Indicator** Project: All Format: MI_BufferSecurityType When this command is executed from within a batch buffer (i.e., is a "chained" batch buffer command), this field is IGNORED and the next buffer in the chain inherits the initial buffer's security characteristics. [DevSNB] When **Per-Process GTT Enable** is set, it is assumed that all code is in a secure environment, independent of address space. Under this condition, this bit only specifies the address space (GGTT or PPGTT). All commands are executed "as-is"

Value	Name	Description	Project
0h	GGTT	This batch buffer is secure and will be accessed via the GGTT.	All
1h	PPGTT	This batch buffer will always be accessed via the PPGTT	All

	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total - Bias
1	31:2	**Batch Buffer Start Address** Project: All Address: GraphicsAddress[31:2] Surface Type: BatchBuffer This field specifies Bits 31:2 of the starting address of the batch buffer.
	1:0	**Reserved** Project: All Format: MBZ

1.2.7.1 Command Access of Privileged Memory

Memory space mapped through the global GTT is considered “privileged” memory. Commands that have the capability of accessing both privileged and unprivileged (PPGTT space) memory will contain a bit that, if set, will attempt a “privileged” access through the GGTT rather than an unprivileged access through the context-local PPGTT.

“User mode” command buffers should not be able to access privileged memory under any circumstances. These command buffers will be issued by the kernel mode driver with the batch buffer’s **Buffer Security** Indicator set to “non-secure”. Commands in such a batch buffer are not allowed to access privileged memory. The commands in these buffers are supplied by the user mode driver and will not be validated by the kernel mode driver. For a batch buffer marked as non-secure if **Per-Process Virtual Address Space is set**, the command buffer fetches are generated using the PPGTT space.

“Kernel mode” command buffers are allowed to access privileged memory. The batch buffers Buffer Security indicator is set to “secure” in this case. In some of the commands that access memory in a secure batch buffer, a bit is provided in the command to steer the access to Per process or Global virtual space. Secure batch buffers are executed from the global GTT.

Commands in ring buffers and commands in batch buffers that are marked as secure (by the kernel mode driver) are allowed to access both privileged and unprivileged memory and may choose on a command-by-command basis.

Table 1-3. GGTT and PPGTT Usage by Command

Command	Address	Allowed Access
MI_BATCH_BUFFER_START*	Command Address	Selectable
MI_DISPLAY_FLIP	Display Buffer Base	GGTT Only
MI_STORE_DATA_IMM*	Storage Address	Selectable
MI_STORE_DATA_INDEX**	Storage Offset	Selectable
MI_STORE_REGISTER_MEM*	Storage Address	Selectable
MI_SEMAPHORE_MBOX	Semaphore Address	Selectable
PIPE_CONTROL	STDW Address	Selectable

*Command has a GGTT/PPGTT selector added to it vs. previous products.

**Added bit allows offset to apply to global HW Status Page or PP HW Status Page found in context image.

1.2.8 MI_CLFLUSH

MI_CLFLUSH

Project:	[DevSNB]	**Length Bias:**	2
Engine:	**Render**		

Flushes out the page given in the command out to system memory. This command is specific to the render engine. This command is *not* privileged.

The MI_CLFLUSH will generare zero-length cycles which look like zero-length writes which are dropped in SuperQ to optimize performance. To get the MI_CLFLUSH to the ring/LLC, the zero-length optimization should be disabled. S/W should disable the optimization via SGCM bit18 (set to "0") before pushing CLFLUSHs to CS ring and re-enable it after observing the storeDW post CLFLUSHs are complete.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 27h Store DW MI_CLFLUSH Format: OpCode
	22	**Use Global GTT** Project: All This bit will be ignored and treated as if clear when executing from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit *must* be '1' if the **Per Process GTT Enable** bit is clear.
	21:8	**Reserved** Project: All Format: MBZ

Value	Name	Description	Project
0h	Per Process Graphics Address		All
1h	Global Graphics Address	This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.	All

MI_CLFLUSH		
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:6	**Page Base Address** Project: All Address: GraphicsAddress[31:0] 4KB aligned Page Address which software requires hardware to flush to DRAM.
	11:0	**Reserved** Project: All Format: MBZ
2..n	31:0	**DW Representing ½ Cache Line** Project: All MBZ. The information given to hardware is the DW itself, not the contents. Hardware uses the DW count of the command to determine the offset from the base to flush out. The offset is ½ cache line (8 DW = 1HW) granular so for a full page, the command will need 4096 bytes / 4 bytes per DW / 8 DW per HW = 128 DW. Note that this is not possible given the 5:0 DW length. Software must split up the DWs with multiple MI_CLFLUSH commands. Example seen below 1st MI_CLFLUSH: address 11:0 = 0, header 5:0 = 0x3FE (62 - 1/2 CL) 2st MI_CLFLUSH: address 11:0 = 62*32, header 5:0 = 0x3FE (62 - 1/2 CL) 3st MI_CLFLUSH: address 11:0 = 62*64, header 5:0 = 0x3FE (62 - 1/2 CL) 4st MI_CLFLUSH: address 11:0 = 62*96, header 5:0 = 0x3FE (62 - 1/2 CL) ... etc.. until all requested cachelines are flushed.

1.2.9 MI_DISPLAY_FLIP

MI_DISPLAY_FLIP			
Project:	All	**Length Bias:**	2
Engine:	Render		

The MI_DISPLAY_FLIP command is used to request a specific display plane to switch (flip) to display a new buffer. The buffer is specified with a starting address and pitch. The tiled attribute of the buffer start address is programmed as part of the packet.

The operation this command performs is also known as a "display flip request" operation – in that the flip operation itself will occur at some point in the future. This command specifies when the flip operation is to occur: either synchronously with vertical retrace to avoid tearing artifacts (possibly on a future frame), or asynchronously (as soon as possible) to minimize rendering stalls at the cost of tearing artifacts.

Programming Notes:

1. This command simply requests a display flip operation -- command execution then continues normally. There is no guarantee that the flip (even if asynchronous) will occur prior to subsequent commands being executed. (Note that completion of the MI_FLUSH command does not guarantee that outstanding flip operations have completed). The MI_WAIT_FOR_EVENT command can be used to provide this synchronization – by pausing command execution until a pending flip has actually completed. This synchronization can also be performed by use of the Display Flip Pending hardware status.

2. After a display flip operation is requested, software is responsible for initiating any required synchronization with subsequent buffer clear or rendering operations. For multi-buffering (e.g., double buffering) operations, this will typically require updating SURFACE_STATE or the binding table to change the rendering (back) buffer. In addition, prior to any subsequent clear or rendering operations, software must typically ensure that the new rendering buffer is not actively being displayed. Again, the MI_WAIT_FOR_EVENT command or Display Flip Pending hardware status can be used to provide this synchronization.

3. The display buffer command uses the X and Y offset for the tiled buffers from the Display Interface registers. Software is allowed to change the offset via the MMIO interface irrespective of the flip commands enqueued in the command stream. For tiled buffers, the display subsystem uses the X and Y offset in generation of the final request to memory. The offset is always updated on the next vblank for both Synchronous and Asynch Flips. It is not necessary to have a flip enqueued to update the X and Y offset

4. The display buffer command uses the linear dword offset for the linear buffers from the Display Interface registers. Software is allowed to change the offset via the MMIO interface irrespective of the flip commands enqueued in the command stream. For linear buffers, the display subsystem uses the dword offset in generation of the final request to memory.

   - For synchronous flips the offset is updated on the next vblank. It is not necessary to have a sync flip enqueued to update the dword offset.

MI_DISPLAY_FLIP

- Linear memory does not support asynchronous flips

5. DWord 3 (Left Eye Display Buffer Base Address) must not be set with synchronous flips or asynchronous flips. It is only allowed to be sent with stereo 3D flips

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 14h MI_DISPLAY_FLIP Format: OpCode
	22	**Async Flip Indicator** Project: All Format: Enable This bit should always be set if DW2 [1:0] == '01' (async flip). This field is required due to HW limitations. This bit is used by the render pipe while DW2 is used by the display hardware.
	21:20	**Display (Plane) Select** Project: [DevSNB] Format: U2 FormatDesc This field selects which display plane is to perform the flip operation.

Value	Name	Description	Project
0h	Display Plane A		All
1h	Display Plane B		All
2h	Display Sprite A		All
3h	Display Sprite B		All

	19:8	**Reserved** Project: Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2 *For Synchronous Flips and Asynchronous Flips,*, this field must be programmed to 1h for a total length of 3.
1	31	**Reserved**
	30:16	**Reserved** Project: All Format: MBZ

MI_DISPLAY_FLIP

	15:6	**Display Buffer Pitch** Project: All Default Value: 0h DefaultVaueDesc Format: U10 *For Synchronous Flips and Stereo 3D Flips only*, this field specifies the 64-byte aligned pitch of the new display buffer. For Asynchronous Flips, this parameter is programmed so that all the flips in a flip chain should maintain the same pitch as programmed with the last synchronous flip or stereo 3D flip or direct thru mmio.
	5:1	**Reserved** Project: All Format: MBZ
	0	**Tile Parameter** Project: [DevSNB+] Default Value: 0h DefaultVaueDesc Format: Enable For Asynchronous Flips, this parameter cannot be changed. All the flips in a flip chain should maintain the same tile parameter as programmed with the last synchronous flip or direct thru mmio.

Value	Name	Description	Project
0h	Linear	For Syncronous Flips Only	All
1h	Tiled X		All

Programming Notes
Performing a synchronous or asynchronous flip will drop any previous synchronous flip that has not yet completed.

2	31:12	**Display Buffer Base Address** Project: All Address: GraphicsAddress[31:12] This field specifies Bits 31:12 of the Graphics Address of the new display buffer. In stereo 3D mode this is the right eye base address.

Programming Notes
• The Display buffer must reside completely in Main Memory • This address is always translated via the *global* (rather than per-process) GTT

	11:3	**Reserved** Project: All Format: MBZ
	2	**Reserved**

MI_DISPLAY_FLIP

	1:0	**Flip Type** Project: All Default Value: 00b Synchronous flip This field specifies whether the flip operation should be performed asynchronously to vertical retrace.

Value	Name	Description	Project
00b	Sync Flip	The flip will occur during the vertical blanking interval – thus avoiding any tearing artifacts.	All
01b	Async Flip	The flip will occur "as soon as possible" – and may exhibit tearing artifacts	All

Programming Notes

- The **Display Buffer Pitch and Tile parameter** fields cannot be changed for asynchronous flips (i.e., the new buffer must have the same pitch/tile format as the previous buffer).
- **Asynch flips are Supported on X-Tiled Frame buffers only.**
- For Asynch Flips the Buffers used must be 32KB aligned.
- Asynch flips Supported on Display Planes A and B and C only

3	31:12	**Reserved**
	11:0	**Reserved** Project: All Format: MBZ

1.2.10 MI_FLUSH

MI_FLUSH			
Project:	All	**Length Bias:**	1
Engine:	**Render**		

The MI_FLUSH command is used to perform an internal "flush" operation. The parser pauses on an internal flush until all drawing engines have completed any pending operations and the read caches are invalidated including the texture cache accessed via the Sampler or the data port. In addition, this command can also be used to:

1. Flush any dirty data in the Render Cache to memory. This is done by default, however this can be inhibited.
2. Invalidate the state and command cache.

Usage note: After this command is completed and followed by a Store DWord-type command, CPU access to graphics memory will be coherent (assuming the Render Cache flush is not inhibited). This command is specific to the render engine. Other engines use MI_FLUSH_DW

Note that if no post-sync operation is enabled for Flush completion, a register write to DE scratch space will be generated by command streamer. Scratch space description is given in DE Bspecs.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 04h MI_FLUSH Format: OpCode
	22:7	**Reserved** Project: All Format: MBZ
	6	**Protected memory Enable** Project: All Format: Enable After completion of the flush, the hardware will limit all access to the Protected Content Memory. Only command streamer initiated cacheable writes are allowed to non-PCM memory.
	5	**Indirect State Pointers Disable** Project: All Format: Disable At the completion of the flush, the indirect state pointers in the hardware will be considered as invalid ie the indirect pointers will not be restored for the context.
	4	**Generic Media State Clear** Project: All Format: Disable If set, all generic media state context information will not be included with the next context save, assuming no new state is initiated after the flush. If clear, the generic media state context save state will not be affected. An MI_FLUSH with this bit set should be issued once all the Media Objects that will be processed by a given persistent root thread have been issued or when an MI_SET_CONTEXT switching from a generic media context to a 3D context completes. When using MI_SET_CONTEXT, once state is programmed, it will be saved and restarted as part of any context each time that context is saved/restored until an MI_FLUSH with this bit set is issued in that context.

MI_FLUSH

	3	**Reserved** Project: All Format:
	2	**Render Cache Flush Inhibit** Project: All Format: Boolean If set, the Render Cache is not flushed as part of the processing of this command.

Value	Name	Description	Project
0h	Flush	Flush the Render Cache	All
1h	Don't Flush	Do not flush the Render Cache	All

	1	**State/Instruction Cache Invalidate** Project: All Format: Boolean If set, Invalidates the State and Instruction Cache

Value	Name	Description	Project
0h	Don't Invalidate	Leave State/Instruction Cache unaffected	All
1h	Invalidate	Invalidate State/Instruction Cache	All

	0	**Reserved** Project: All Format: MBZ

1.2.11 MI_LOAD_REGISTER_IMM

MI_LOAD_REGISTER_IMM			
Project:	All	**Length Bias:**	2
Engine:	**Render**		

The MI_LOAD_REGISTER_IMM command requests a write of up to a DWord constant supplied in the command to the specified Register Offset (i.e., offset into Memory-Mapped Register Range).

Programming Notes:

- A stalling flush must be sent down pipeline before issuing this command
- The behavior of this command is controlled by Dword 3, Bit 8 (**Disable Register Access**) of the RINGBUF register. If this command is disallowed then the command stream converts it to a NOOP.
- If this command is executed from a BB then the behavior of this command is controlled by Dword 0, Bit 8 (**Security Indicator**) of the BATCH_BUFFER_START Command. If the batch buffer is insecure then the command stream converts this command to a NOOP. Note that the corresponding ring buffer must allow a register update for this command to execute.
- To ensure this command gets executed before upcoming commands in the ring, either a stalling pipeControl should be sent after this command, or MMIO 0x20C0 bit 7 should be set to 1.
- When base address of 0x180000 is added to the Register Offset, when executed will result in updating of the register in the other GT in GTB mode of operation then the GT from which this instruction is executed. When this instruction is executed by Command Streamer with COREID-0 will result in updating the register in GT with COREID-1 and vice versa, when base address of 0x180000 is added to the register offset.

The following addresses should NOT be used for LRIs

1. 0x8800 - 0x88FF
2. >= 0xC0000

Limited LRI cycles to the Display Engine 0x40000-0xBFFFF) are allowed, but must be spaced to allow only one pending at a time. This can be done by issuing an SRM to the same address immediately after each LRI.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 22h MI_LOAD_REGISTER_IMM Format: OpCode

MI_LOAD_REGISTER_IMM		
	22:12	**Reserved** Project: All Format: MBZ
	11:8	**Byte Write Disables** Format: Enable[4] Bit 8 corresponds to Data DWord [7:0] Range Must specify a valid register write operation If [11:8] is '1111', then this command will behave as a NOOP. Otherwise, the value is forwarded to the destination register.
	7:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:2	**Register Offset** Format: U30 Address: MmioAddress[31:2] This field specifies bits [31:2] of the offset into the Memory Mapped Register Range (i.e., this field specifies a DWord offset). When base address of 0x180000 is added to the Register Offset, when executed will result in updating of the register in the other GT in GTB mode of operation then the GT from which this instruction is executed. When this instruction is executed by Command Streamer with COREID-0 will result in updating the register in GT with COREID-1 and vice versa, when base address of 0x180000 is added to the register offset.
	1:0	**Reserved** Project: All Format: MBZ
2	31:0	**Data DWord** Mask: Bytes Write Disables Format: U32 This field specifies the DWord value to be written to the targeted location.

1.2.12 MI_NOOP

MI_NOOP

Project:	All	**Length Bias:**	1
Engine:	**Render**		

The MI_NOOP command basically performs a “no operation” in the command stream and is typically used to pad the command stream (e.g., in order to pad out a batch buffer to a QWord boundary). However, there is one minor (optional) function this command can perform – a 22-bit value can be loaded into the MI NOPID register. This provides a general-purpose command stream tagging ("breadcrumb") mechanism (e.g., to provide sequencing information for a subsequent breakpoint interrupt).

Performance Note:

On previous products, the process time to execute a NOP command is min of 6 clock cycles.

On **[DevSNB],** the NOP process time is reduced to 1 clock. One example usage of the improved NOP throughput is for some multi-pass media application whereas some unwanted media object commands are replaced by MI_NOOP without repacking the commands in a batch buffer.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0h MI_NOOP Format: OpCode
	22	**Identification Number Register Write Enable** Project: All Format: Enable This field enables the value in the Identification Number field to be written into the MI NOPID register. If disabled, that register is unmodified – making this command an effective “no operation” function.

Value	Name	Description	Project
0h	Disable	Do not write the NOP_ID register.	All
1h	Enable	Write the NOP_ID register.	All

31:0	**Identification Number** Project: All Format: U22 This field contains a 22-bit number which can be written to the MI NOPID register.	

1.2.13 Surface Probing

These commands are only valid when the “Surface Fault Enable” bit is set in the GFX_MODE register

1.2.14 MI_REPORT_HEAD

MI_REPORT_HEAD			
Project:	All	**Length Bias:**	1
Engine:	**Render**		

The MI_REPORT_HEAD command causes the Head Pointer value of the active ring buffer to be written to a cacheable (snooped) system memory location.

The location written is relative to the address programmed in the Hardware Status Page Address Register.

Programming Notes:

- This command must not be executed from a Batch Buffer.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 07h MI_REPORT_HEAD Format: OpCode
	22:0	**Reserved** Project: All Format: MBZ

1.2.15 MI_SEMAPHORE_MBOX

MI_SEMAPHORE_MBOX			
Project:	[DevSNB]	**Length Bias:**	2
Engine:	**Render**		

This command is provided as alternative to MI_SEMAPHORE to provide mailbox-type semaphores where there is no update of the semaphore by the checking process (the consumer). Single-bit compare-and-update semantics are also provided. In either case, atomic access of semaphores need not be guaranteed by hardware as with the previous command. This command should eventually supersede the previous command.

Synchronization between contexts (especially between contexts running on 2 different engines) is provided by the MI_SEMAPHORE_MBOX command. Note that contexts attempting to synchronize in this fashion must be able to access a common memory location. This means the contexts must share the same virtual address space (have the same page directory), must have a common physical page mapped into both of their respective address spaces or the semaphore commands must be executing from a secure batch buffer or directly from a ring with the **Use Global GTT** bit set such that they are "privileged" and will use the (always shared) global GTT.

MI_SEMAPHORE with the **Update Semaphore** bit set (and the **Compare Semaphore** bit clear) implements the *Signal* command, while the *Wait* command is indicated by **Compare Semaphore** being set. Note that *Wait* can cause a context switch. *Signal* increments unconditionally.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 16h MI_SEMAPHORE_MBOX Format: OpCode
	22	**Use Global GTT** Project: All Format: U32 If set, this command will use the global GTT to translate the **Semaphore Address** and this command must be executing from a privileged (secure) batch buffer. If clear, the PPGTT will be used to translate the **Semaphore Address**. This bit will be ignored (and treated as if clear) if this command is executed from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer or directly from a ring buffer.
	21	**Update Semaphore** Project: All Format: U32 If set, the value from the **Semaphore Data Dword** is written to memory. If **Compare Semaphore** is also set, the semaphore is not updated if the semaphore comparison fails. If clear, the data at **Semaphore Address** is not changed.

MI_SEMAPHORE_MBOX		
	20	**Compare Semaphore** Project: All Format: U32 If set, the value from the **Semaphore Data Dword** is compared to the value from the **Semaphore Address** in memory. If the value at **Semaphore Address is greater than the Semaphore Data Dword**, execution is continued from the current command buffer. If clear, no comparison takes place. **Update Semaphore** *must* be set in this case.
	19	**Reserved** Project: All Format: MBZ
	18	**Compare Register** Project: All Format: Compare Type If set, data in MMIO register will be used for compare. If clear, data in memory will be used for compare.
	17:16	**Register Select** Project: All Format: Register Select If compare register is set in bit[18], this filed indicate which register will be used. 0: VCS register (RVSYNC) 1: [Reserved] 2: BCS regiser (RBSYNC) 3. Use General Register Select
	15:14	**Reserved** Project: All Format: MBZ
	13:8	**Reserved** Project: **All** Format:
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:0	**Semaphore Data Dword** Project: All Format: U32 Data dword to compare/update memory. The Data dword is supplied by software to control execution of the command buffer. If the compare is enabled and the data at **Semaphore Address** is greater than this dword, the execution of the command buffer continues.
2	31:2	**PointerBitFieldName/MMIO Register Address** Project: All Address: GraphicsVirtualAddress[31:2] Surface Type: Semaphore if Compare Register bit[18] is cleared, this field is the Graphics Memory Address of the 32 bit value for the semaphore. If Compare Register bit[18] is set, this field is the MMIO address of the register for the semaphore.
	1:0	**Reserved** Project: All Format: MBZ

1.2.16 MI_SET_CONTEXT

MI_SET_CONTEXT			
Project:	All	**Length Bias:**	2
Engine:	**Render**		

The MI_SET_CONTEXT command is used to specify the *logical* context associated with the hardware context. A logical context is an area in memory used to store hardware context information, and the context is referenced via a 2KB-aligned pointer. If the (new) logical context is different (i.e., at a different memory address), the device will proceed to save the current HW context values to the current logical context address, and then restore (load) the new logical context by reading the context from the new address and loading it into the hardware context state. If the logical context address specified in this command matches the current logical context address, this command is effectively treated as a NOP.

This command also includes some controls over the context save/restore process. It is specific to the render engine

- The **Force Restore** bit can be used to refresh the on-chip device state from the same memory address if the indirect state buffers have been modified.
- The **Restore Inhibit** bit can be used to prevent the new context from being loaded at all. This **must** be used to prevent an uninitialized context from being loaded. Once software has initialized a context (by setting all state variables to initial values via commands), the context can then be stored and restored normally.
- This command is legal only if **Per-Process Virtual Address Space** in the GFX_MODE register is re*set*.
- This command needs to be always followed by a single MI_NOOP instruction to workaround a Gen4 silicon issue.
- When switching from a generic media context to a 3D context, the generic media state must be cleared via the *Generic Media State Clear* bit 16 in PIPE_CONTROL (or bit 4 in MI_FLUSH) before saving 3D context.
- [DevSNB] If **Flush TLB invalidation Mode** is *enabled* it's the driver's responsibility to invalidate the TLBs at least once after the previous context switch after any GTT mappings changed (including new GTT entries). This can be done by a pipelined PIPE_CONTROL with TLB inv bit set immediately before MI_SET_CONTEXT.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 18h MI_SET_CONTEXT Format: OpCode
	22:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2

MI_SET_CONTEXT		
1	31:12	**Logical Context Address** Project: All Address: GraphicsAddress[31:12] Surface Type: Logical Context This field contains the 4KB-aligned physical address of the Logical Context that is to be loaded into the hardware context. If this address is equal to the CCID register associated with the current ring, no load will occur. Prior to loading this new context, the device will save the existing context as required. After the context switch operation completes, this address will be loaded into the associated CCID register. **[DevSNB] This field needs to be 4KB aligned virtual address.**
	11:10	**Reserved** Project: All Format: MBZ
	9	**Reserved** Format: MBZ
	8	**Reserved, Must be 1** Project: All Format: Must Be One
	7:5	**Reserved** Project: All Format: MBZ
	4	Reserved
	3	**Extended State Save Enable** Project: {DevSNB] Format: U32 If set, the extended state identified in the Logical Context Data section of the Memory Data Formats chapter is saved as part of switching away from this logical context. This bit will be stored in the associated CCID register to control the context save operation when switching away from this context (as part of a subsequent MI_SET_CONTEXT command). This bit must be '1' when RS2 power state is enabled (via MCHBAR, offset 0x11B8)
	3	**Reserved** Project: Format:
	2	**Extended State Restore Enable** Project: {DevSNB] Format: U32 If set, the extended state identified in the Logical Context Data section of the Memory Data Formats chapter is loaded (or restored) as part of switching to this logical context. This method can be used to restore things such as filter coefficients using the indirect state restore followed by a restore of the extended logical context data. This bit affects the switch (if required) to the context specified in **Logical Context Address**. This bit will also be stored in the associated CCID register to control a subsequent context save operation when switching to this context (as part of a subsequent ring buffer switch). This bit must be '1' when RS2 power state is enabled (via MCHBAR, offset 0x11B8)
	2	**Reserved** Project: Format:

MI_SET_CONTEXT		
	1	**Force Restore** Project: All Format: U32 When switching to this logical context a comparison between Logical Context Address and the contests of the CCID register is performed. Normally, matching addresses prevent a context restore from occurring; however, when this bit is set a context restore is forced to occur. This bit cannot be set with Restore Inhibit. **Note:** This bit is not saved in the associated CCID register. It only affects the processing of this command.
	0	**Restore Inhibit** Project: All Format: U32 If set, the restore of the HW context from the logical context specified by **Logical Context Address** is inhibited (i.e., the existing HW context values are maintained). This bit must be used to prevent the loading of an uninitialized logical context. If clear, the context switch proceeds normally. This bit cannot be set with Force Restore. **Note:** This bit is not saved in the associated CCID register. It only affects the processing of this command.

1.2.17 MI_STORE_DATA_IMM

MI_STORE_DATA_IMM

Project:	All	**Length Bias:**	2
Engine:	**Render**		

The MI_STORE_DATA_IMM command requests a write of the QWord constant supplied in the packet to the specified Memory Address. As the write targets a System Memory Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes:

This command should not be used within a "non-secure" batch buffer to access global virtual space. Doing so will cause the command parser to perform the write with byte enables turned off. This command can be used within ring buffers and/or "secure" batch buffers.

This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll un-cached memory or device registers).

This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete "eventually", there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 20h MI_STORE_DATA_IMM Format: OpCode
	22	**Use Global GTT** Project: All This bit will be ignored and treated as if clear when executing from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit *must* be '1' if the **Per Process GTT Enable** bit is clear.

Value	Name	Description	Project
0h	Reserved		
1h	Global Graphics Address	This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.	All

DWord	Bit	Description
	21:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) = 2 for DWord, 3 for QWord Format: =n Total Length - 2
1	31:0	**Reserved** Project: All Format: MBZ

MI_STORE_DATA_IMM		
2	31:2	**Address** Project: All Address: GraphicsAddress[31:2] Surface Type: U32(2) This field specifies Bits 31:2 of the Address where the DWord will be stored. As the store address must be DWord-aligned, Bits 1:0 of that address MBZ. This address must be 8B aligned for a store "QW" command.
	1	**Reserved** Project: All Format: MBZ
	0	Reserved Project: Format:
3	31:0	**Data DWord 0** Project: All Format: U32 This field specifies the DWord value to be written to the targeted location. For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).
4	31:0	**Data DWord 1** Project: All Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

1.2.18 MI_STORE_DATA_INDEX

MI_STORE_DATA_INDEX			
Project:	All	**Length Bias:**	2
Engine:	**Render**		

The MI_STORE_DATA_INDEX command requests a write of the data constant supplied in the packet to the specified offset from the System Address defined by the Hardware Status Page Address Register. As the write targets a System Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes:

Use of this command with an invalid or uninitialized value in the Hardware Status Page Address Register is UNDEFINED.

This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll uncached memory or device registers).

This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete "eventually", there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode

MI_STORE_DATA_INDEX		
	28:23	**MI Command Opcode** Default Value: 21h MI_STORE_DATA_INDEX Format: OpCode
	22	**Reserved** Project: Format:
	21	**Use Per-Process Hardware Status Page** Project: All If this bit is set, this command will index into the per-process hardware status page at offset 28K from the LRCA. If clear, the Global Hardware Status Page will be indexed. This bit will be ignored and treated as set if this command is executed from within a non-secure batch buffer. This but must always be '0'
	20:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) = 1 for DWord, 2 for QWord Format: =n Total Length - 2
1	31:12	**Reserved** Project: All Format: MBZ
	11:2	**Offset** Project: All Format: U10 zero-based DWord offset into the HW status page. Address: HardwareStatusPageOffset[11:2] Surface Type: U32 Range [16, 1023] This field specifies the offset (into the hardware status page) to which the data will be written. Note that the first few DWords of this status page are reserved for special-purpose data storage – targeting these reserved locations via this command is UNDEFINED. This address must be 8B aligned for a store "QW" command.
	1:0	**Reserved** Project: All Format: MBZ
2	31:0	**Data DWord 0** Project: All Format: U32 This field specifies the DWord value to be written to the targeted location. For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).
3	31:0	**Data DWord 1** Project: All Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

1.2.19 MI_STORE_REGISTER_MEM

MI_STORE_REGISTER_MEM			
Project:	All	**Length Bias:**	2
Engine:	**Render**		

The MI_STORE_REGISTER_MEM command requests a register read from a specified memory mapped register location in the device and store of that DWord to memory. The register address is specified along with the command to perform the read.
Programming Notes:

The command temporarily halts command execution.

The memory address for the write is snooped on the host bus.

This command should not be used within a "non-secure" batch buffer to access global virtual space. Doing so will cause the command parser to perform the write with byte enables turned off. This command can be used within ring buffers and/or "secure" batch buffers.

This command will cause undefined data to be written to memory if given register addresses for the PGTBL_CTL_0 or FENCE registers

The following addresses should NOT be used for SRMs

1. 0x8800 - 0x88FF
2. >= 0x40000

The only exception is an SRM cycle to 0x40000-0xBFFFF when used as part of the LRI read-after-write requirement.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 24h MI_STORE_REGISTER_MEM Format: OpCode

MI_STORE_REGISTER_MEM

	22	**Use Global GTT** Project: All This bit will be ignored and treated as if clear when executing from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit *must* be '1' if the **Per Process GTT Enable** bit is clear.

Value	Name	Description	Project
0h	Reserved		
1h	Global Graphics Address	This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.	All

	21	**Reserved** Format: MBZ
	20:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:26	**Reserved** Project: All Format: MBZ
	25:2	**Register Address** Project: All Address: MMIO Address[25:2] Surface Type: MMIO Register This field specifies Bits 25:2 of the Register offset the DWord will be read from. As the register address must be DWord-aligned, Bits 1:0 of that address MBZ.

Programming Notes	Project
Storing a VGA register is not permitted and will store an UNDEFINED value.	All
The values of PGTBL_CTL0 or any of the FENCE registers cannot be stored to memory; UNDEFINED values will be written to memory if the addresses of these registers are specified.	All

	1:0	**Reserved** Project: All Format: MBZ

MI_STORE_REGISTER_MEM		
2	31:2	**Memory Address** Project: All Address: GraphicsAddress[31:2] Surface Type: MMIO Register This field specifies the address of the memory location where the register value specified in the DWord above will be written. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[31:2] for a DWord register
	1	**Reserved** Project: All Format: MBZ
	0	Reserved

1.2.20 MI_SUSPEND_FLUSH

MI_SUSPEND_FLUSH			
Project: All **Engine:** **Render**		**Length Bias:**	1
Blocks MMIO sync flush or any flushes related to VT-d while enabled.			
DWord	**Bit**	**Description**	
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode	
	28:23	**MI Command Opcode** Default Value: 0Bh MI_SUSPEND_FLUSH Format: OpCode	
	22:1	**Reserved** Project: All Format: MBZ	
	0	**Suspend Flush** Project: All Default Value: 0h DefaultVaueDesc Format: Enable FormatDesc This field suspends flush due to sync flush or implicit flush generated during VTD enable, disable and IOTLB invalidation.	

Value	Name	Description	Project
0h	Disable		All
1h	Enable		All

1.2.21 MI_UPDATE_GTT

MI_UPDATE_GTT			
Project:	All	**Length Bias:**	2
Engine:	Render		

The MI_UPDATE_GTT command is used to update GTT page table entries in a coherent manner and at a predictable place in the command flow.

An MI_FLUSH should be placed before this command, since work associated with preceding commands that are still in the pipeline may be referencing GTT entries that will be changed by its execution. The flush will also invalidate TLBs and read caches that may become invalid as a result of the changed GTT entries. MI_FLUSH is not required if it can be guaranteed that the pipeline is free of any work that relies on changing GTT entries (such as MI_UPDATE_GTT contained in a paging DMA buffer that is doing only update/mapping activities and no rendering).

This is a privileged command. This command will be converted to a no-op and an error flagged if it is executed from within a non-secure batch buffer.

Note that MI_UPDATE_GTT is mainly for the pages that are strictly used by GT. If driver chooses to update the CPU used pages thru MI_UPDATE_GTT, it needs to write to MMIO address x101008 (any value) to ensure system agent TLBs are invalidated before the new pages can be used.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 23h MI_UPDATE_GTT Format: OpCode
	22	**Use Global GTT** Project: All
	21:8	**Reserved** Project: All Format: MBZ

Value	Name	Description	Project
0h	Per Process Graphics Address	This command will use the Per Process GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.	All
1h	Global Graphics Address	This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.	All

MI_UPDATE_GTT		
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:12	**Entry Address** Project: All Address: GraphicsAddress[31:12] This field simply holds the DW offset of the first table entry to be modified. Note that one or more of the upper bits may need to be 0, i.e., for a 2G aperture, bit 31 MBZ.
	11:0	**Reserved** Project: All Format: MBZ
2..n	31:0	**Entry Data** Project: All Format: Table Entry This Dword becomes the new page table entry. See PPGTT/Global GTT Table Entries (PTEs) in Memory Interface Registers.

1.2.22 MI_USER_INTERRUPT

MI_USER_INTERRUPT			
Project:	All	**Length Bias:**	1
Engine:	**Render**		
The MI_USER_INTERRUPT command is used to generate a User Interrupt condition. The parser will continue parsing after processing this command. See User Interrupt.			
DWord	**Bit**	**Description**	
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode	
	28:23	**MI Command Opcode** Default Value: 02h MI_USER_INTERRUPT Format: OpCode	
	22:0	**Reserved** Project: All Format: MBZ	

1.2.23 MI_WAIT_FOR_EVENT

MI_WAIT_FOR_EVENT			
Project:	All	**Length Bias:**	1
Engine:	Render		

The MI_WAIT_FOR_EVENT command is used to pause command stream processing until a specific event occurs or while a specific condition exists. See Wait Events/Conditions, Device Programming Interface in *MI Functions*. Only one event/condition can be specified -- specifying multiple events is UNDEFINED.

The effect of the wait operation depends on the source of the command. Once parsed, the parser will halt (and suspend command arbitration) until the event/condition occurs. Note that if a specified condition does not exist (the condition code is inactive) at the time the parser executes this command, the parser proceeds, treating this command as a no-operation.

If CSunit is waiting for V-blank or flip done, HW can go into RC1/RC6 state.

Software must disable MI_WAIT_FOR_EVENT RC6 entry via RC_PSMI_CTRL if MI_WAIT_FOR_EVENT is parsed in a batch buffer with the following attributes set:

- batch buffer in PPGTT space (labeled “non-secure” in command)
- CB^2 batch buffer

MI_NOOP setting NOP register (or any other benign command) must be set after MI_WAIT_FOR_EVENT under the following conditions

- Back-to-back MI_WAIT_FOR_EVENT commands
- MI_WAIT_FOR_EVENT is the last command before head = tail

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 03h MI_WAIT_FOR_EVENT Format: OpCode
	22:20	**Reserved** Project: Format: MBZ
	22	**Reserved** Project: Format:
	21	**Reserved** Project: Format:

MI_WAIT_FOR_EVENT

	20	**Reserved** Project: Format:
	19:16	**Condition Code Wait Select** Project: All This field enables a wait for the duration that the corresponding condition code is active. These enable select one of 15 condition codes in the EXCC register, that cause the parser to wait until that condition-code in the EXCC is cleared.

Value	Name	Description	Project
0h	Not Enabled	Condition Code Wait not enabled	All
1h-5h	Enabled	Condition Code select enabled; selects one of 5 codes, 0 – 4	All
6h-15h	Reserved		All

Programming Notes
Note that not all condition codes are implemented. The parser operation is UNDEFINED if an unimplemented condition code is selected by this field.

	15:14	**Reserved** Project: Format: MBZ
	13	**Display Pipe B H Blank Wait Enable** Project: All Format: Enable This field enables a wait until the start of next Display Pipe B “Horizontal Blank” event occurs. This event is defined as the start of the next Display B Horizontal blank period. Note that this can cause a wait for up to a line. See Horizontal Blank Event in the Device Programming Interface chapter of *MI Functions*.
	12	**Reserved** Project: All Format: MBZ
	11	**Display Pipe B Vertical Blank Wait Enable** Project: All Format: U32 This field enables a wait until the next Display Pipe B “Vertical Blank” event occurs. This event is defined as the start of the next Display Pipe B vertical blank period. Note that this can cause a wait for up to an entire refresh period. See Vertical Blank Event (See *Programming Interface*).
	10	**Display Sprite B Flip Pending Wait Enable** Project: All Format: Enable This field enables a wait for the duration of a Display Sprite B “Flip Pending” condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

MI_WAIT_FOR_EVENT		
	9	**Display Plane B Flip Pending Wait Enable** Project: All Format: Enable This field enables a wait for the duration of a Display Plane B “Flip Pending” condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	8	**Display Pipe B Scan Line Wait Enable** Project: All Format: Enable This field enables a wait while a Display Pipe B “Scan Line” condition exists. This condition is defined as the the start of the scan line specified in the Pipe B Display Scan Line Count Range Compare Register.
	7:6	**Reserved** Project: All Format: MBZ
	5	**Display Pipe A H Blank Wait Enable** Project: All Format: U32 This field enables a wait until the start of next Display Pipe A “Horizontal Blank” event occurs. This event is defined as the start of the next Display A Horizontal blank period. Note that this can cause a wait for up to a line.
	4	**Reserved** Project: All Format: MBZ
	3	**Display Pipe A Vertical Blank Wait Enable** Project: All Format: Enable This field enables a wait until the next Display Pipe A “Vertical Blank” event occurs. This event is defined as the start of the next Display A vertical blank period. Note that this can cause a wait for up to an entire refresh period.
	2	**Display Sprite A Flip Pending Wait Enable** Project: All Format: Enable This field enables a wait for the duration of a Display Sprite A “Flip Pending” condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of *MI Functions*.
	1	**Display Plane A Flip Pending Wait Enable** Project: All Format: Enable This field enables a wait for the duration of a Display Plane A “Flip Pending” condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of *MI Functions*.

MI_WAIT_FOR_EVENT						
	0	**Display Pipe A Scan Line Wait Enable**	Project:	All	Format:	Enable
		This field enables a wait while a Display Pipe A “Scan Line” condition exists. This condition is defined as the the start of the scan line specified in the Pipe A Display Scan Line Count Range Compare Register. See Scan Line Event in the Device Programming Interface chapter of *MI Functions*.				

Revision History

Revision Number	Description	Revision Date
1.0	First 2011 Opensource edition	May 2011

§§

Intel® OpenSource HD Graphics Programmer's Reference Manual (PRM)

Volume 1 Part 4: Graphics Core – Video Codec Engine (SandyBridge)

For the 2011 Intel Core Processor Family

May 2011

Revision 1.0

NOTICE:

This document contains information on products in the design phase of development, and Intel reserves the right to add or remove product features at any time, with or without changes to this open source documentation.

Doc Ref #: IHD-OS-V1 Pt4 – 05 11

Creative Commons License

You are free to Share — to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any
way that suggests that they endorse you or your use of the work).

No Derivative Works. You may not alter, transform, or build upon this work.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The SandyBridge chipset family, Havendale/Auburndale chipset family, Intel® 965 Express Chipset Family, Intel® G35 Express Chipset, and Intel® 965GMx Chipset Mobile Family Graphics Controller may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel® sales office or your distributor to obtain the latest specifications and before placing your product order. I2C is a two-wire communications bus/protocol developed by Philips. SMBus is a subset of the I2C bus/protocol and was developed by Intel®. Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

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

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

Copyright © 2011, Intel Corporation. All rights reserved.

Contents

1. Video Codec Engine Command Streamer 4

1.1 Registers for Video Codec 4
1.1.1 Introduction 4
1.1.2 Virtual Memory Control 4
1.1.3 Context Submission [DevSNB] 11
1.1.4 VCS_RINGBUF—Ring Buffer Registers 13
1.1.5 Watchdog Timer Registers 16
1.1.6 Interrupt Control Registers 17
1.1.7 Logical Context Support 24
1.1.8 Registers in MFC Pipe [DevSNB+] 26
1.1.9 Registers in Media Engine 33
1.2 Memory Interface Commands for Video Codec Engine 35
1.2.1 Introduction 35
1.2.2 MI_ARB_CHECK 35
1.2.3 MI_ARB_ON_OFF 36
1.2.4 MI_BATCH_BUFFER_END 37
1.2.5 MI_CONDITIONAL_BATCH_BUFFER_END 37
1.2.6 MI_BATCH_BUFFER_START 39
1.2.7 MI_FLUSH_DW 41
1.2.8 MI_LOAD_REGISTER_IMM 44
1.2.9 MI_NOOP 45
1.2.10 MI_REPORT_HEAD 46
1.2.11 MI_SEMAPHORE_MBOX 47
1.2.12 MI_STORE_REGISTER_MEM 49
1.2.13 MI_STORE_DATA_IMM 51
1.2.14 MI_STORE_DATA_INDEX 53
1.2.15 MI_SUSPEND_FLUSH 55
1.2.16 MI_USER_INTERRUPT 55
1.2.17 MI_UPDATE_GTT 56
1.2.18 MI_WAIT_FOR_EVENT 57

1. Video Codec Engine Command Streamer

For [DevSNB+], full decode pipeline as well as encode pipeline are implemented in VCE.

VCE has its own command streamer and operates completely independently of the render (3D/Media) pipeline command streamer.

1.1 Registers for Video Codec

1.1.1 Introduction

This command streamer supports a completely independent set of registers. Only a subset of the MI Registers is supported for this 2nd command streamer. The effort is to keep the registers at the same offset as the render command streamer registers. The base of the registers for the video decode engine will be defined per project, the offsets will be maintained.

Project	Base Address Value for the memory interface register offset for the Bit Stream Command Stream
DevSNB+	0x10000 eg: The Ring buffer tail pointer will be 0x10000 + 0x2030

1.1.2 Virtual Memory Control

MFX engine supports a 2-level mapping scheme for PPGTT, consisting of a first-level page directory containing page table base addresses, and the page tables themselves on the 2nd level, consisting of page addresses.

1.1.2.1 VCS_PP_DIR_BASE – Page Directory Base Register

VCS_PP_DIR_BASE – Page Directory Base Register	
Register Type:	MMIO_CS
Address Offset:	12228h
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

This register contains the offset into the GGTT where the (current context's) PPGTT page directory begins. This register is restored with context. The Page Directory Base Address is set by SW only by modifying the value of this register in the context image such that the new value is restored the next time the context runs. A write via MMIO to this register triggers the render pipe to fetch all PDs.

Programming Note: The **MBC Driver Boot Enable** bit in **MBCTL** register must be set *before* this register is written to upon boot up (including S3 exit)

Bit	Description
30:16	**Page Directory Base Offset** Project: All Default Value: 0h Format: U15 Range [0,GGTT Size in cachelines - 1] Contains the cacheline (64-byte) offset into the GGTT where the page directory begins.
15:1	**Reserved** Project: All Format: MBZ
0	**PD Load Busy** Project: DevSNB+ Format: Valid This is a read-only field that indicates if the page directories are currently being fetched and loaded.

1.1.2.2 VCS_PP_DCLV – PPGTT Directory Cacheline Valid Register

VCS_PP_DCLV – PPGTT Directory Cacheline Valid Register	
Register Type:	MMIO_CS
Address Offset:	12220h
Project:	All
Default Value:	0h
Access:	R/W
Size (in bits):	64

This register controls update of the on-chip PPGTT Directory Cache during a context restore. Bits that are set will trigger the load of the corresponding 16 directory entry group. This register is restored with context (prior to restoring the on-chip directory cache itself). This register is also restored when switching to a context whose LRCA matches the current CCID if the **Force PD Restore** bit is set in the context descriptor.

The context image of this register must be updated and maintained by SW; SW should not normally need to read this register.

This register can also effectively be used to limit the size of a processes' virtual address space. Any access by a process that requires a PD entry in a set that is not enabled in this register will cause a fatal error, and no fetch of the PD entry will be attempted

Bit	Description
63:32	**Reserved** Project: All Format: MBZ
31:0	**PPGTT Directory Cache Restore [1..32] 16 entries** Project: All Format: Array:Enable If set, the [1st..32nd] 16 entries of the directory cache are considered valid and will be brought in on context restore. If clear, these entries are considered invalid and fetch of these entries will not be attempted.

The field below needs to go in some register to enable PPGTT, either in GAC MMIO or VCS MMIO.

1	**Per-Process GTT Enable** Project: DevSNB+ Format: Enable If set, PPGTT support in hardware is enabled. This bit *must* be set if runlist enable is set. Setting this bit also allows support for big pages (32k)

1.1.2.3 VCS_MI_MODE — Mode Register for Software Interface

VCS_MI_MODE — Mode Register for Software Interface

Register Type:

Address Offset: 1209Ch–1209Fh

Project:

Default Value: 0000 0200h

Access: Read/Write

Size (in bits): 32 bits

The MI_MODE register contains information that controls software interface aspects of the command parser

Bit	Description
31:16	**Masks:** A "1" in a bit in this field allows the modification of the corresponding bit in Bits 15:0
15	**Suspend Flush** **Project:** **All** **Mask:** MMIO(0x209c)#31 **Value** / **Name** / **Description** / **Project** **Oh** / No Delay / HW will not delay flush, this bit will get cleared by MI_SUSPEND_FLUSH as well / **All** **1h** / Delay Flush / HW will delay the flush because of sync flush or VTD regimes until reset, this bit will get set by MI_SUSPEND_FLUSH as well / **All**
14:12	**Reserved** Read/Write
11	**Invalidate UHPTR enable:** If bit set H/W clears the valid bit of BCS_UHPTR (4134h, bit 0) when current active head pointer is equal to UHPTR.
10	**Reserved** Read/Write
9	**Ring Idle (Read Only Status bit)** 0 = Parser not Idle 1 = Parser Idle ***Writes to this bit are not allowed.***
8	**Stop Ring** 0 = Normal Operation. 1 = Parser is turned off. Software must set this bit to force the Ring and Command Parser to Idle. Software must read a "1" in Ring Idle bit after setting this bit to ensure that the hardware is idle. ***Software must clear this bit for Ring to resume normal operation.***
7:3	**Reserved** Read/Write

VCS_MI_MODE — Mode Register for Software Interface	
Register Type:	
Address Offset:	1209Ch–1209Fh
Project:	
Default Value:	0000 0200h
Access:	Read/Write
Size (in bits):	32 bits
The MI_MODE register contains information that controls software interface aspects of the command parser	

Bit	Description
2	**MI_ARB_ON_OFF Privileged Attribute Enable [DevSNB+]** If set, the MI_ARB_ON_OFF command will be treated as a privileged command. That is, if executed in a non-secure batch buffer, hardware will convert it to a NOOP. If clear, hardware will execute at all times. Note that this register cannot be changed in the UMD.
1:0	**Reserved** Read/Write

1.1.2.4 VCS_INSTPM—Instruction Parser Mode Register

Address Offset: 120C0h–120C3h

Default Value: 0000 0000h

Access: Read/Write

Size: 32 bits

The VCS_INSTPM register is used to control the operation of the VCS Instruction Parser. Certain classes of instructions can be disabled (ignored) – often useful for detecting performance bottlenecks. Also, “Synchronizing Flush” operations can be initiated – useful for ensuring the completion (vs. only parsing) of rendering instructions.

Programming Notes:

- All Reserved bits are implemented.

Bit	Description
31:16	**Masks:** These bits serve as write enables for bits 15:0. If this register is written with any of these bits clear the corresponding bit in the field 15:0 will not be modified. Reading these bits always returns 0s.
15:10	Reserved: MBZ
9	**Reserved** Project: Format:

Bit	Description
8:7	Reserved: MBZ
6	**Memory Sync Enable:** This set, this bit allows the video decode engine to write out the data from the local caches to memory. **[DevSNB+]** This bit is *not* persistent. S/W must define this bit each time a sync flush is requested
5	**Sync Flush Enable:** This field is used to request a Sync Flush operation. The device will automatically clear this bit before completing the operation. See Sync Flush (*Programming Environment*). **Programming Note:** • The command parser must be stopped prior to issuing this command by setting the **Stop Ring** bit in register **BCS_MI_MODE**. Only after observing **Ring Idle** set in **BCS_MI_MODE** can a Sync Flush be issued by setting this bit. Once this bit becomes clear again, indicating flush complete, the command parser is re-enabled by clearing **Stop Ring**. Format = Enable (cleared by HW) [DevSNB+] When using MI_SUSPEND_FLUSH, this bit cannot be relied on as an indicator of sync flush complete. Instead, driver must wait until head == tail
4:0	Reserved: MBZ

1.1.2.5 VCS_NOPID — NOP Identification Register

Address Offset: 12094h–12097h

Default Value: 0000 0000h

Access: Read Only

Size: 32 bits

The BCS_NOPID register contains the Noop Identification value specified by the last MI_NOOP instruction that enabled this register to be updated.

Bit	Description
31:22	Reserved: MBZ
21:0	**Identification Number:** This field contains the 22-bit Noop Identification value specified by the last MI_NOOP instruction that enabled this field to be updated.

1.1.2.6 VBSYNC – Video/Blitter Semaphore Sync Register

VBSYNC – Video/Blitter Semaphore Sync Register	
Register Type:	MMIO_VCS
Address Offset:	12040h
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Trusted Type:	1
This register is written by BCS, read by VCS.	
Bit	**Description**
31:0	**Semaphore Data** Semaphore data for synchronization between video codec engine and blitter engine..

1.1.2.7 VRSYNC – Video/Render Semaphore Sync Register

VRSYNC – Video/Render Semaphore Sync Register	
Register Type:	MMIO_VCS
Address Offset:	12044h
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Trusted Type:	1
This register is written by CS, read by VCS.	
Bit	**Description**
31:0	**Semaphore Data** Semaphore data for synchronization between video codec engine and render engine.

1.1.2.8 GAC_MODE — Mode Register for GAC

Address Offset: 120A0h–120A3h

Default Value: 0000 0000h

Access: Read/Write

Size: 32 bits

The GAC_MODE register contains information that controls configurations in the GAC.

Bit	Description
31:16	**Masks:** A "1" in a bit in this field allows the modification of the corresponding bit in Bits 15:0
15:0	**Reserved** Read/Write

1.1.3 Context Submission [DevSNB]

1.1.3.1 VCS_RCCID—Ring Buffer Current Context ID Register

Address Offset: 12190h–12197h

Default Value: 00 00 00 00h

Access: Read/Write

Size: 32 bits

This register contains the current "ring context ID" associated with the ring buffer.

Programming Notes:

- The current context registers must not be written directly (via MMIO). The RCCID register should only be updated indirectly from RNCID.

Bit	Description
63:0	See Context Descriptor for VCS

1.1.3.2 VCS_RNCID—Ring Buffer Next Context ID Register

Address Offset: 12198h–1219fh

Default Value: 00 00 00 00h

Access: Read/Write

Size: 64 bits

This register contains the *next* "ring context ID" associated with the ring buffer.

Programming Notes:

- The current context (RCCID) register can be updated indirectly from this register on a context switch event. Note that this can only be triggered when arbitration is enabled or if the current context runs dry (head pointer becomes equal to tail pointer).

Bit	Description
63:0	See Context Descriptor for VCS

1.1.3.3 Context Status

A context switch interrupt will be sent anytime a context switch occurs. This is documented in the "GPU Overview" volume, "Memory Data Formats" chapter. A status DW for the context that was just switched away from will be written to the Context Status Buffer in the Global Hardware Status Page. The status contains the context ID and the reason for the context switch. Note that since there will have been no running contexts when the very first (after reset) context is submitted, the Context ID in the first Context Status DWord will be UNDEFINED.

Table 1-1. Format of Context Status Dword

Bit	Description
31:12	**Context ID.** Contains the context ID copied from the submitted context.
11:8	Reserved: MBZ
7	**Media watch dog timer expired** cause the context switch
6	Reserved: MBZ
5	Reserved: MBZ
4	**Ring Buffer Becoming Empty** Caused context to Switch.
3	Reserved: MBZ
2	Reserved: MBZ
1	**Waiting on a Semaphore** Caused Context to Switch.
0	Reserved: MBZ

When SW services a context switch interrupt, it should read the Context Status Buffer beginning where it left off reading the last time it serviced a context switch interrupt. It should read up through the **Last Written Status Offset**, which is also recorded in the Context Status Buffer. The status DWs can be examined to determine which contexts were switched out between context interrupt service intervals, and why.

Table 1-2. Number of Context Status Entries in Memory

Device	Number of Status Entries
DevSNB	12 (DW) Entries

Status Dwords are written out to the Context Status Buffer at incrementing addresses. The Context Status Buffer has a limited size and simply wraps around to the beginning when the end is reached.

The Context Status Buffer fits into a single cacheline so that the whole buffer will be read from memory at once if the driver performs a cacheable read.

Table 1-3. Format of the Context Status Buffer

DW	Description
15	**Last Written Status Offset.** This Dword is written on every context switch with the (pre-increment) value of the **Context Status Buffer Pointer Register**. The lower 4 bits increment for every status Dword write; the upper 28 bits are always 0. The lowest 4 bits indicate which of the Context Status Dwords was just written.
14-12	Reserved: MBZ
11-0	**Context Status Dwords.** A circular buffer of context status DWs. As each context is switched away from, its status is written here at ascending DWs as indicated by the **Last Written Status Offset**. Once DW 11 has been written, the pointer wraps around so that the next status will be written at DW0. Format = ContextStatusDW

1.1.4 VCS_RINGBUF—Ring Buffer Registers

Address Offset: 12030h – 1203Fh: Ring Buffer:

offset 0h = _TAIL

offset 4h = _HEAD

offset 8h = _START

offset Ch = _CTL

Default Value: 0000 0000h

Access: Read/32 bit Write Only

Size: 4 DWords / Ring Buffer

These registers are used to define and operate the “ring buffer” mechanism which can be used to pass instructions to the command interface. The buffer itself is located in a linear memory region. The ring buffer is defined by a 4 Dword register set that includes starting address, length, head offset, tail offset, and control information. Refer to the *Programming Interface* chapter for a detailed description of the parameters specified in this ring buffer register set, restrictions on the placement of ring buffer memory, arbitration rules, and in how the ring buffer can be used to pass instructions.

Ring Buffer Head and Tail Offsets must be properly programmed before it is enabled. A Ring Buffer can be enabled when empty.

The format of the Ring Buffer register set follows:

<table>
<tr><th>DWord Offset</th><th>Bit</th><th>Description</th></tr>
<tr><td rowspan="3">0</td><td>31:21</td><td>Reserved: MBZ</td></tr>
<tr><td>20:3</td><td>

Tail Offset: This field is written by software to specify where the valid instructions placed in the ring buffer end. The value written points to the QWord *past* the last valid QWord of instructions. In other words, it can be defined as the *next* QWord that software will write instructions into. Software must write subsequent instructions to QWords following the Tail Offset, possibly wrapping around to the top of the buffer (i.e., software can't skip around within the buffer). Note that all DWords prior to the location indicated by the **Tail Offset** must contain valid instruction data – which may require instruction padding by software. See **Head Offset** for more information.

Format = U18 QWord Offset

[DevSNB] Every tail move must follow the sequence below

<table>
<tr><th>MMIO action</th><th>Comment</th></tr>
<tr><td>Write 0x12050 = 0x00010001</td><td>Disable MFX pipe from claiming idle</td></tr>
<tr><td>Write 0x12198 = 0x00000000</td><td>Benign active cycle that will wake up MFX pipe (if currently idle)</td></tr>
<tr><td>Poll for 0x12050[3] = 0</td><td>Make sure MFX pipe is out of idle. Very unlikely will need more than 1 read</td></tr>
<tr><td>Write 0x12030 = <new tail ptr value></td><td></td></tr>
<tr><td>Write 0x12050 = 0x00010000</td><td>Let VCS claim MFX pipe idle again</td></tr>
</table>

</td></tr>
<tr><td>2:0</td><td>Reserved: MBZ</td></tr>
<tr><td rowspan="2">1</td><td>31:21</td><td>

Wrap Count: This field is incremented by 1 whenever the **Head Offset** wraps from the end of the buffer back to the start (i.e., whenever it wraps back to 0). Appending this field to the **Head Offset** field effectively creates a virtual 4GB Head "Pointer" which can be used as a tag associated with instructions placed in a ring buffer. The Wrap Count itself will wrap to 0 upon overflow.

Format = U11 count of ring buffer wraps

</td></tr>
<tr><td>20:2</td><td>

Head Offset: This field indicates the offset of the *next* instruction DWord to be parsed. Software will initialize this field to select the first DWord to be parsed once the RB is enabled. (Writing the Head Offset while the RB is enabled is UNDEFINED). Subsequently, the device will increment this offset as it executes instructions – until it reaches the QWord specified by the **Tail Offset**. At this point the ring buffer is considered "empty".

Programming Notes:

A RB can be enabled empty or containing some number of valid instructions.

Format = U19 DWord Offset

</td></tr>
<tr><td></td><td>1:0</td><td>Reserved: MBZ</td></tr>
</table>

DWord Offset	Bit	Description
2	31:12	**Starting Address:** This field specifies Bits 31:12 of the 4KB-aligned starting Graphics Address of the ring buffer. All ring buffer pages must map to Main Memory (uncached) pages. Ring Buffer addresses are always translated through the global GTT. Per-process address space can only be used via a batch buffer. Format: Graphics Address Bits 31:12
	11:0	Reserved: MBZ
3	31:21	Reserved: MBZ
	20:12	**Buffer Length:** This field is written by SW to specify the length of the ring buffer in 4 KB Pages. Format = U9 in 4 KB pages – 1 Range = [0 = 1 page = 4 KB, 1FFh = 512 pages = 2 MB]
	11	**RBWait** Indicates that this ring has executed a WAIT_FOR_EVENT instruction and is currently waiting. Software can write a "1" to clear this bit, write of "0" has no effect. When the RB is waiting for an event and this bit is cleared, the wait will be terminated and the RB will be returned to arbitration.
	10	**Semaphore Wait** Indicates that this ring has executed a MI_SEMAPHORE_MBOX instruction with register compare and is currently waiting. Software can write a "1" to clear this bit, write of "0" has no effect. When the RB is waiting for the compare to meet and this bit is cleared, the wait will be terminated and the RB will be returned to arbitration.
	9	**Reserved:** MBZ
	8	**Disable Register Accesses:** 0 = Ring is allowed to access (read or write) MMIO space. 1 = Ring is not allowed to <u>write</u> MMIO space. Ring *is* allowed to <u>read</u> registers.
	7:3	Reserved: MBZ
	2:1	**Automatic Report Head Pointer:** This field is written by software to control the automatic "reporting" (write) of this ring buffer's "Head Pointer" register (register DWord 1) to the corresponding location within the Hardware Status Page. Automatic reporting can either be disabled or enabled at 4KB, 64KB or 128KB boundaries within the ring buffer. Format = 0: MI_AUTOREPORT_OFF – Automatic reporting disabled 1: MI_AUTOREPORT_64KB – Report every 16 pages (64KB) 2: MI_AUTOREPORT_4KB – Report every page (4KB) 3: MI_AUTOREPORT_128KB – Report every 32 pages (128KB) When the **Per-Process Virtual Address Space** bit is set and automatic head reporting is desired, this field must be set to option 2 since the ring buffer will be only 16KB in size. The head pointer will be reported to the head pointer location in the PP HW Status Page when it passes each 4KB page boundary. When the above-mentioned bit is set, reporting will behave just as on the prior devices (as documented above), and option 2 is not legal.
	0	**Ring Buffer Enable:** This field is used to enable or disable this ring buffer. It can be enabled or disabled regardless of whether there are valid instructions pending. Format = Enable

1.1.4.1 VCS_UHPTR — Pending Head Pointer Register

Address Offset: 12134h–12137h

Default Value: 0000 0000h

Access: Read/Write

Size: 32 bits

Bit	Description
31:3	**Head Pointer Address**: This register represents the GFX address offset where execution should continue in the ring buffer following execution of an MI_ARB_CHECK command. Format = MI_Graphics_Offset
2:1	Reserved: MBZ
0	**Head Pointer Valid:** 1 = Indicates that there is an updated head pointer programmed in this register 0 = No valid updated head pointer register, resume execution at the current location in the ring buffer This bit is set by the software to request a pre-emption. It is reset by hardware when an MI_ARB_CHECK command is parsed by the command streamer. The hardware uses the head pointer programmed in this register at the time the reset is generated.

1.1.5 Watchdog Timer Registers

1.1.5.1 VCS_CNTR—Counter for the bit stream decode engine

Address Offset: 12178h–1217Bh

Default Value: FFFF FFFFh

Access: Read/Write

Size: 32 bits

Bit	Description
31:0	**Count Value**: Writing a Zero value to this register starts the counting. Writing a Value of FFFF FFFF to this counter stops the counter

1.1.5.2 VCS_THRSH—Threshold for the counter of bit stream decode engine

Address Offset: 1217Ch–1217Fh

Default Value: 00150000h

Access: Read/Write

Size: 32 bits

Bit	Description
31:0	**Threshold Value**: The value in this register reflects the number of clocks the bit stream decode engine is expected to run. If the value is exceeded the counter is reset and an interrupt may be enabled in the device.

1.1.6 Interrupt Control Registers

The Interrupt Control Registers described below all share the same bit definition. The bit definition is as follows:

Table 1-4. Bit Definition for Interrupt Control Registers

Bit	Description
31:21	**Reserved. MBZ:** These bits may be assigned to interrupts on future products/steppings.
20	**Context Switch Interrupt:** Set when a context switch has just occurred. **Per-Process Virtual Address Space bit** needs to be set for this interrupt to occur**.**
19	**Page Fault:** This bit is set whenever there is a pending PPGTT (page or directory) fault.
18	**Timeout Counter Expired:** Set when the VCS timeout counter has reached the timeout thresh-hold value.
17	**Reserved**
16	**MI_FLUSH_DW Notify Interrupt:** The Pipe Control packet (Fences) specified in *3D pipeline* document may optionally generate an Interrupt. The Store QW associated with a fence is completed ahead of the interrupt.
15	**Video Command Parser Master Error:** When this status bit is set, it indicates that the hardware has detected an error. It is set by the device upon an error condition and cleared by a CPU write of a one to the appropriate bit contained in the Error ID register followed by a write of a one to this bit in the IIR. Further information on the source of the error comes from the "Error Status Register" which along with the "Error Mask Register" determine which error conditions will cause the error status bit to be set and the interrupt to occur. **Page Table Error:** Indicates a page table error. **Instruction Parser Error**: The Video Instruction Parser encounters an error while parsing an instruction.
14	**Sync Status:** This bit is set when the Instruction Parser completes a flush with the sync enable bit active in the INSTPM register. The event will happen after all the MFX engines are flushed. The HW Status DWord write resulting from this event will cause the CPU's view of graphics memory to be coherent as well (flush and invalidate the MFX cache). It is the driver's responsibility to clear this bit before the next sync flush with HWSP write enabled

Bit	Description
13	Reserved
12	**Video Command Parser User Interrupt:** This status bit is set when an MI_USER_INTERRUPT instruction is executed on the Video Command Parser. Note that instruction execution is not halted and proceeds normally. A mechanism such as an MI_STORE_DATA instruction is required to associate a particular meaning to a user interrupt.
11:0	**Reserved:** MBZ

The following table specifies the settings of interrupt bits stored upon a "Hardware Status Write" due to ISR changes:

Bit	Interrupt Bit	ISR bit Reporting via Hardware Status Write (when unmasked via HWSTAM)
8	**Context Switch Interrupt:** Set when a context switch has just occurred.	Not supported to be unmasked
7	**Page Fault:** This bit is set whenever there is a pending PPGTT (page or directory) fault.	Set when event occurs, cleared when event cleared
6	**Media Decode Pipeline Counter Exceeded Notify Interrupt:** The counter threshold for the execution of the media pipeline is exceeded. Driver needs to attempt hang recovery.	Not supported to be unmasked
5	**Reserved**	
4	MI_FLUSH_DW packet - Notify Enable	0
3	Master Error	Set when error occurs, cleared when error cleared
2	Sync Status	Set every SyncFlush Event
1	Reserved	
0	User Interrupt	0

1.1.6.1 HWSTAM — Hardware Status Mask Register

Hardware Status Mask Register	
Register Type:	MMIO_VCS
Address Offset:	12098h
Project:	All
Default Value:	**FFFF FFFFh**
Access:	R/W, RO for Reserved Control bits
Size (in bits):	32
Trusted Type:	1

The HWSTAM register has the same format as the Interrupt Control Registers. The bits in this register are "mask" bits that prevent the corresponding bits in the Interrupt Status Register from generating a "Hardware Status Write" (PCI write cycle). Any unmasked interrupt bit (HWSTAM bit set to 0) will allow the Interrupt Status Register to be written to the ISR location (within the memory page specified by the Hardware Status Page Address Register) when that Interrupt Status Register bit changes state.

Programming Notes:

- To write the interrupt to the HWSP, the corresponding IMR bit must also be clear (enabled).
- At most 1 bit can be unmasked at any given time.

Bit	Description
31:0	**Hardware Status Mask Register** Project: All Default Value: FFFFFFFFh DefaultVaueDesc Format: Array of Masks refer to Table 4-4 in Interrupt Control Register section for bit definitions

1.1.6.2 IMR—Interrupt Mask Register

IMR—Interrupt Mask Register	
Register Type:	MMIO_VCS
Address Offset:	120A8h
Project:	All
Default Value:	FFFF FFFFh
Access:	R/W
Size (in bits):	32

The IMR register is used by software to control which Interrupt Status Register bits are “masked” or “unmasked”. “Unmasked” bits will be reported in the IIR, possibly triggering a CPU interrupt, and will persist in the IIR until cleared by software. “Masked” bits will not be reported in the IIR and therefore cannot generate CPU interrupts.

Bit	Description
31:0	**Interrupt Mask Bits** Project: All Default Value: FFFF FFFFh Format: Array of interrupt mask bits — Refer to Interrupt Control Register section for bit definitions This field contains a bit mask which selects which interrupt bits (from the ISR) are reported in the IIR.

Value	Name	Description	Project
0h	Not Masked	Will be reported in the IIR	All
1h	Masked	Will not be reported in the IIR	All

1.1.6.3 Hardware-Detected Error Bit Definitions (for EIR, EMR, ESR)

This section defines the Hardware-Detected Error bit definitions and ordering that is common to the EIR, EMR and ESR registers. The EMR selects which error conditions (bits) in the ESR are reported in the EIR. Any bit set in the EIR will cause the Master Error bit in the ISR to be set. EIR bits will remain set until the appropriate bit(s) in the EIR is cleared by writing the appropriate EIR bits with '1'(except for the unrecoverable bits described below).

The following table describes the Hardware-Detected Error bits:

Table 1-5. Hardware-Detected Error Bits

Bit	Description
15:1	Reserved: MBZ
0	**Instruction Error:** This bit is set when the Renderer Instruction Parser detects an error while parsing an instruction. Instruction errors include: 1) Client ID value (Bits 31:29 of the Header) is not supported (only MI, 2D and 3D are supported). 2) Defeatured MI Instruction Opcodes: 1: Instruction Error detected **Programming Note:** This error indications can not be cleared except by reset (i.e., it is a fatal error).

1.1.6.3.1 EIR — Error Identity Register

EIR — Error Identity Register					
Register Type:	MMIO_VCS				
Address Offset:	120B0h				
Project:	All				
Default Value:	0000 0000h				
Access:	R/WC				
Size (in bits):	32				
The EIR register contains the persistent values of Hardware-Detected Error Condition bits. Any bit set in this register will cause the Master Error bit in the ISR to be set. The EIR register is also used by software to clear detected errors (by writing a '1' to the appropriate bit(s) except for the unrecoverable bits described).					
Bit	**Description**				
31:16	**Reserved**	Project:	All	Format:	MBZ

EIR — Error Identity Register

15:0	**Error Identity Bits** Project: All Default Value: 0h Format: Array of Error condition bits See Table 1 5. Hardware-Detected Error Bits This register contains the persistent values of ESR error status bits that are unmasked via the EMR register. (See **Error! Reference source not found.**). The logical OR of all (defined) bits in this register is reported in the Master Error bit of the Interrupt Status Register. In order to clear an error condition, software must first clear the error by writing a '1' to the appropriate bit(s) in this field. If required, software should then proceed to clear the Master Error bit of the IIR.

Value	Name	Description	Project
1h	Error occurred	Error occurred	All

Programming Notes	Project
Writing a '1' to a set bit will cause that error condition to be cleared. However, the Page Table Error bit (Bit 4) can not be cleared except by reset (i.e., it is a fatal error).	All

1.1.6.3.2 EMR—Error Mask Register

EMR—Error Mask Register					
Register Type: MMIO_VCS **Address Offset:** 120B4h **Project:** All **Default Value:** FFFF FFFFh **Access:** R/W **Size (in bits):** 32					
The EMR register is used by software to control which Error Status Register bits are "masked" or "unmasked". "Unmasked" bits will be reported in the EIR, thus setting the Master Error ISR bit and possibly triggering a CPU interrupt, and will persist in the EIR until cleared by software. "Masked" bits will not be reported in the EIR and therefore cannot generate Master Error conditions or CPU interrupts.					
Bit	**Description**				
31:16	**Reserved** Project: All Format: MBZ				
15:0	**Error Mask Bits** Project: All Default Value: FFFF FFFFh Format: Array of error condition mask bits See Table 1 5. Hardware-Detected Error Bits This register contains a bit mask that selects which error condition bits (from the ESR) are reported in the EIR.				
	Value	**Name**	**Description**		**Project**
	0h	Not Masked	Will be reported in the EIR		All
	1h	Masked	Will not be reported in the EIR		All

1.1.6.3.3 ESR—Error Status Register

ESR—Error Status Register					
Register Type: MMIO_VCS **Address Offset:** 120B8h **Project:** All **Default Value:** 0000 0000h **Access:** RO **Size (in bits):** 32					
The ESR register contains the current values of all Hardware-Detected Error condition bits (these are all by definition "persistent"). The EMR register selects which of these error conditions are reported in the persistent EIR (i.e., set bits must be cleared by software) and thereby causing a Master Error interrupt condition to be reported in the ISR.					
Bit	**Description**				
31:16	**Reserved** Project: All Format: MBZ				

ESR—Error Status Register	
15:0	**Error Status Bits** Project: All Default Value: 0h Format: Array of error condition bits See Table 1 5. Hardware-Detected Error Bits This register contains the non-persistent values of all hardware-detected error condition bits. **Value** / **Name** / **Description** / **Project** 1h / Error Condition Detected / Error Condition detected / All

1.1.7 Logical Context Support

1.1.7.1 VCS_BB_ADDR—Batch Buffer Head Pointer Register

Address Offset: 012140h–012147h

Default Value: 0000 0000 0000 0000h

Access: Read-Only

Size: 64 bits

This register contains the current QWord Graphics Memory Address of the last-initiated batch buffer.

Bit	Description
63:32	Reserved: MBZ
31:3	**Batch Buffer Head Pointer:** This field specifies the QWord-aligned Graphics Memory Address where the last initiated Batch Buffer is currently fetching commands. If no batch buffer is currently active, the Valid bit will be 0 and this field will be meaningless. .
2:1	Reserved: MBZ
0	**Valid:** 1 = Batch buffer Valid 0 = Batch buffer Invalid

1.1.7.2 VCS_BB_STATE — Batch Buffer State Register

VCS_BB_STATE – Batch Buffer State Register

Register Type:	MMIO_VCS
Address Offset:	12110h
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

This register contains the attributes of the last batch buffer initiated from the Ring Buffer. These include the security indicator.

This register should *not* be written by software. These fields should only get written by a context restore. Software should always set these fields via the MI_BATCH_BUFFER_START command when initiating a batch buffer.

This register is saved and restored with context.

Bit	Description
31:6	**Reserved** Project: All Format: MBZ
5	**Buffer Security Indicator** Project: All Default Value: 0h Format: MI_BufferSecurityType If set, this batch buffer is non-secure and cannot execute privileged commands nor access privileged (GGTT) memory. It will be accessed via the PPGTT. If clear, this batch buffer is secure and will be accessed via the GGTT. Note: This field reflects the effective security level and may not be the same as the Buffer Security Indicator written using MI_BATCH_BUFFER_START. **Value** / **Name** / **Description** / **Project** 0h / MIBUFFER_SECURE / Located in GGTT memory / All 1h / MIBUFFER_NONSECURE / Located in PPGTT memory / All
4	**Reserved**
3:0	**Reserved** Project: All Format: MBZ

1.1.7.3 VCS_CTXT_SR_CTL — Context Save/Restore Control Register

CTXT_SR_CTL – Context Save/Restore Control Register	
Register Type:	MMIO_VCS
Address Offset:	12114h
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32
This register is saved and restored with context.	
Bit	**Description**
31:1	**Reserved** Project: Format: MBZ
0	**MFX Context Restore Inhibit** Project: Format: U1 ıis is not a true register bit. This bit should be set in the context image of a ring context that is being submitted for the first time. Setting this bit will inhibit the restoring of render context (including extended context if applicable) so that restoring of an uninitialized render context can be prevented. This bit will always be set on a context save (since the render context cannot be uninitialized on context save – it will always contain at least default values.)

1.1.8 Registers in MFC Pipe [DevSNB+]

These registers count for AVC encoder statistics of the parallel Video Codec Engine (VCE). They are saved and restored with context but should not be changed by software during frame processing. These registers are reset to 0 each time when command MFX_PIPE_MODE_SELECT is issued. These registers may be read at any time; however, to obtain a meaningful result, a pipeline flush just prior to reading the registers is necessary in order to synchronize the counts with the primitive stream. These registers can be read to memory through the MI_STORE_REGISTER_MEM command.

1.1.8.1 MFC_VIN_AVD_ERROR_FLAGS — AVC Bitstream Decoding Front-End Parsing Logic Error Report Register

MFC_VIN_AVD_ERROR_FLAGS	
Register Type:	MMIO_VCS
Address Offset:	12400h
Project:	All
Default Value:	00000000h;
Access:	RW
Size (in bits):	32
Trusted Type:	1
The only MMIO write operation is to reset each individual bit of this register to a 0 value, after the error information has been read and/or processed. The driver may choose to read this register in between pictures and video sequence and upon video stream switching. This register is set to 0 at powerup.	

MFC_VIN_AVD_ERROR_FLAGS	
Bit	**Description**
31:0	avd_error_flagsR[31:0] 31:6 -- Reserved 5 – **AVD Error Rewind flag** 4 – **AVD Error Conceal Flag** 3 -- **BSD Premature Completion Error Status Flag** When a BSD Premature Completion error has occurred and the BSDPrematureComplete Error Handling bit in the inline data of the AVC_BSD_OBJECT command is set, this error status flag is set until being cleared by a subsequent MMIO write to this register. 2 -- **MPR Error Status Flag** When a MPR error has occurred and the MPR Error Handling bit in the inline data of the AVC_BSD_OBJECT command is set, this error status flag is set until being cleared by a subsequent MMIO write to this register. 1 -- **VLD Error Status Flag** When a VLD error has occurred and the VLD Error Handling bit in the inline data of the AVC_BSD_OBJECT command is set, this error status flag is set until being cleared by a subsequent MMIO write to this register. 0 -- **BSD Error Status Flag** When a BSD error has occurred and the BSD Error Handling bit in the inline data of the AVC_BSD_OBJECT command is set, this error status flag is set until being cleared by a subsequent MMIO write to this register.

1.1.8.2 MFC_VIN_AVD_ERROR_CNTR — AVC Bitstream Decoding Front-End Parsing Logic Error Counter Report Register

MFC_VIN_AVD_ERROR_CNTR[11:0]	
Register Type:	MMIO_VCS
Address Offset:	12404h
Project:	All
Default Value:	00000000h;
Access:	RW
Size (in bits):	32
Trusted Type:	1

The only MMIO write operation is to reset this register to a 0 value. The driver may choose to read this register in between pictures and video sequence and upon video stream switching. This register is set to 0 at powerup.

Bit	Description
31:0	**avd_error_flagsR[31:0]** : 31:12 -- Reserved 11:0 -- **BSD Error Count --** Increment by 1 when any of the recognized errors (BSD, VLD, MPR and PrematureCompletion) has occurred. Do not wrap around when the maximum count has reached. error_cntR[11:0]

1.1.8.3 MFC_BITSTREAM_BYTECOUNT_SLICE — Bitstream Output Byte Count Per Slice Report Register

MFC_BITSTREAM_BYTECOUNT_SLICE	
Register Type:	MMIO_VCS
Address Offset:	12408h
Project:	[DevSNB+]
Default Value:	00000000h; 00000000h;
Access:	RO
Size (in bits):	32
Trusted Type:	1
This register stores the count of bytes of the bitstream output. This register is part of the context save and restore.	
Bit	**Description**
31:0	**MFC Bitstream Byte Count** Total number of bytes in the bitstream output from the encoder. This count is updated for every time the internal bitstream counter is incremented.

1.1.8.4 MFC_BITSTREAM_SE_BITCOUNT_SLICE — Bitstream Output Bit Count for the last Syntax Element Report Register

MFC_BITSTREAM_SE_BITCOUNT_SLICE	
Register Type:	MMIO_VCS
Address Offset:	1240Ch
Project:	All
Default Value:	00000000h; 00000000h;
Access:	RO
Size (in bits):	32
Trusted Type:	1
This register stores the count of number of bits in the bitstream for the last syntax element before padding. The bit count is before the byte-aligned alignment padding insertion, but includes the stop-one-bit. This register is part of the context save and restore.	
Bit	**Description**
31:0	**FC Bitstream Syntax Element Bit Count** ıtal number of bits in the bitstream output before padding. This count is updated each time the internal counter is incremented.

1.1.8.5 MFC_AVC_CABAC_INSERTION_COUNT — Bitstream Output CABAC Insertion Count Report Register

MFC_AVC_CABAC_INSERTION_COUNT	
Register Type:	MMIO_VCS
Address Offset:	12410h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	RO
Size (in bits):	32
Trusted Type:	1
This register stores the count in bytes of **CABAC ZERO_WORD insertion. It is primarily provided for statistical data gathering.** This register is part of the context save and restore.	
Bit	**Description**
31:0	**MFC AVC Cabac Insertion Count** Total number of bytes in the bitstream output before for the CABAC zero word insertion. This count is updated each time when the insertion count is incremented.

1.1.8.6 MFC_AVC_MINSIZE_PADDING_COUNT — Bitstream Output Minimal Size Padding Count Report Register

MFC_AVC_MINSIZE_PADDING_COUNT	
Register Type:	MMIO_VCS
Address Offset:	12414h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	RO
Size (in bits):	32
Trusted Type:	1
This register stores the count in bytes of **minimal size padding insertion. It is primarily provided for statistical data gathering.** This register is part of the context save and restore.	
Bit	**Description**
31:0	**MFC AVC MinSize Padding Count** Total number of bytes in the bitstream output contributing to minimal size padding operation. This count is updated each time when the padding count is incremented.

1.1.8.7 MFC_IMAGE_STATUS_MASK

MFC_IMAGE_STATUS_MASK	
Register Type:	MMIO
Address Offset:	**12418H**
Project:	DevSNB+
Default Value:	00000000h; 00000000h;
Access:	RO
Size (in bits):	32
Trusted Type:	1
This register stores the image status(flags). This register is part of the context save and restore.	
Bit	**Description**
31:0	Control Mask for dynamic frame repeat

1.1.8.8 MFC_IMAGE_STATUS_CONTROL

MFC_IMAGE_STATUS_CONTROL	
Register Type:	MMIO
Address Offset:	**1241CH**
Project:	DevSNB+
Default Value:	00000000h; 00000000h;
Access:	RO
Size (in bits):	32
Trusted Type:	1
This register stores the suggested data for next frame in multipass. This register is part of the context save and restore.	
Bit	**Description**
31:24	**Reserved**
23:16	suggested slice QP delta value for frame level Rate control. This value can be +ve or -ve
15:2	**Reserved**
1	**Frame Bit count over-run/under-run flag**
0	Max Macroblock conformance flag or Frame Bit count over-run/under-run

1.1.8.9 MFC_BITSTREAM_BYTECOUNT_FRAME — Reported Bitstream Output Byte Count per Frame Register

BITSTREAM_BYTECOUNT_FRAME	
Register Type:	MMIO
Address Offset:	**12420H**
Project:	DevSNB+
Default Value:	00000000h; 00000000h;
Access:	RO
Size (in bits):	32
Trusted Type:	1
This register stores the count of bytes of the bitstream output per frame. This register is part of the context save and restore.	
Bit	**Description**
31:0	**MFC Bitstream Byte Count per Frame** Total number of bytes in the bitstream output per frame from the encoder. This includes header/tail/byte alignment/data bytes/EMU bytes/cabac-zero word insertion/padding insertion. This count is updated for every time the internal bitstream counter is incremented and its reset at image start.

1.1.8.10 MFC_BITSTREAM_SE_BITCOUNT_FRAME — Reported Bitstream Output Bit Count for Syntax Elements Only Register

MFC_BITSTREAM_SE_BITCOUNT_FRAME	
Register Type:	MMIO
Address Offset:	**12424H**
Project:	All
Default Value:	00000000h; 00000000h;
Access:	RO
Size (in bits):	32
Trusted Type:	1
This register stores the count of number of bits in the bitstream due to syntax elements only. This excludes header/ byte alignment /tail/EMU/CABAC-0word/padding bits but includes the stop-one-bit. This register is part of the context save and restore.	
Bit	**Description**
31:0	**MFC Bitstream Syntax Element Only Bit Count** Total number of bits in the bitstream output due to syntax elements only. It includes the data bytes only. This count is updated for every time the internal bitstream counter is incremented and its reset at image start.

1.1.8.11 MFC_AVC_CABAC_BIN_COUNT_FRAME — Reported Bitstream Output CABAC Bin Count Register

MFC_AVC_CABAC_BIN_COUNT_FRAME	
Register Type:	MMIO
Address Offset:	**12428H**
Project:	All
Default Value:	00000000h; 00000000h;
Access:	RO
Size (in bits):	32
Trusted Type:	1
This register stores the count of number of bins per frame. This register is part of the context save and restore.	
Bit	**Description**
31:0	**MFC AVC Cabac Bin Count** Total number of bins in the bitstream output per frame from the encoder. This count is updated for every time the bin counter is incremented and its reset at image start.

1.1.9 Registers in Media Engine

1.1.9.1 Introduction

The register detailed in this chapter is used across the GEN family of products and is an extention to previous projects. However, slight changes may be present (i.e., for features added or removed), or some registers may be removed entirely. These changes are clearly marked within this chapter.

1.1.9.1.1 VCS_HWS_PGA — Hardware Status Page Address Register

VCS_HWS_PGA — Hardware Status Page Address Register

Register Type: MMIO
Address Offset: 04180h
Project: DevSNB+
Default Value: 0000 0000h
Access: R/W
Size (in bits): 32
Trusted Type: 1

This register is used to program the 4 KB-aligned System Memory address of the Hardware Status Page used to report hardware status into (typically cacheable) System Memory. [DevSNB] This address in this register is translated using the Global GTT in memory. The mapping type of the GTT entry determines the snoop nature of the transaction to memory.

Programming Notes
[DevSNB+] If this register is written, a workload must subsequently be dispatched to the video command streamer.

Bit	Description
31:12	**Address** Project: DevSNB+ Security: None Address: GraphicsAddress[31:12] This field is used by SW to specify Bits 31:12 of the 4 KB-aligned System Memory address of the 4 KB page known as the "Hardware Status Page". The Global GTT is used to map this page from the graphics virtual address to physical address **Programming Notes** Notes: If the **Per-Process Virtual Address Space** is set, HW requires that the status page is programmed to allow for the context switch status to be reported
11:1	**Reserved** Project: DevSNB+ Format: MBZ
0	**Translation In Progress** Project: All Format: U1 FormatDesc This field indicates that the translation for the hardware status page from the graphics virtual address to the physical address is pending. Software can use this indicator to prevent updating the status page when there is a pending cycle for translation.

1.2 Memory Interface Commands for Video Codec Engine

1.2.1 Introduction

This chapter describes the formats of the “Memory Interface” commands, including brief descriptions of their use. The functions performed by these commands are discussed fully in the *Memory Interface Functions* Device Programming Environment chapter.

This chapter describes MI Commands for the Video Codec Engine.

The commands detailed in this chapter are used across the later products within the GEN family. However, slight changes may be present in some commands (i.e., for features added or removed), or some commands may be removed entirely. Refer to the *Preface* chapter for details.

1.2.2 MI_ARB_CHECK

The instruction format is:

MI_ARB_CHECK

Project:	All	**Length Bias:**	1
Engine:	**Video**		

The MI_ARB_CHECK instruction is used with the UHPTR register. This instruction can be used to pre-empt the current execution of the ring buffer. Note that the valid bit in the UHPTR register needs to be set for the command streamer to be pre-empted.

Programming Note:

This instruction can be placed only in a ring buffer, never in a batch buffer.

DWord	Bit	Description
0	31:29	**MI Instruction Type** Default Value: 0h MI_INSTRUCTION Format: OpCode
	28:23	**MI Instruction Opcode** Default Value: 05h MI_ARB_CHECK Format: OpCode
	22:0	**Reserved** Project: All Format: MBZ

1.2.3 MI_ARB_ON_OFF

MI_ARB_ON_OFF			
Project:	DevSNB+	**Length Bias:**	1
Engine:	**Video**		

The MI_ARB_ON_OFF instruction is used to disable/enable context switching. This command will also prevent a switch in the case of running out of commands. This will effectively hang the device if allowed to occur while arbitration is off (context switching is disabled.)

This command should always be used as an off-on pair with the sequence of instructions to be protected from context switch between MI_ARB_OFF and MI_ARB_ON.

This is a privileged command only if the MI_ARB_ON_OFF privileged bit is set in the VCS_MI_MODE register; it will not be effective (will be converted to a no-op) if executed from within a non-secure batch buffer. This command can only be issued when **Per-Process Virtual Address Space** is set; if the bit is set it will be converted to NOOP.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 08h MI_ARB_ON_OFF Format: OpCode
	22:1	**Reserved** Project: All Format: MBZ
	0	**Arbitration Enable** Format: Enable This field enables or disables context switches due to pre-emption **Value** **Name** 0h Disabled 1h Enabled

1.2.4 MI_BATCH_BUFFER_END

The MI_BATCH_BUFFER_END command format follows:

MI_BATCH_BUFFER_END			
Project: All **Engine:** **Video**		**Length Bias:**	1
The MI_BATCH_BUFFER_END command is used to terminate the execution of commands stored in a *batch buffer* initiated using a MI_BATCH_BUFFER_START command.			
DWord	**Bit**	**Description**	
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode	
	28:23	**MI Command Opcode** Default Value: 0Ah MI_BATCH+_BUFFER_END Format: OpCode	
	22:0	**Reserved** Project: All Format: MBZ	

1.2.5 MI_CONDITIONAL_BATCH_BUFFER_END

MI_CONDITIONAL_BATCH_BUFFER_END			
Project: DevSNB+ **Engine:** **Video**		**Length Bias:**	2
The MI_BATCH_BUFFER_END command is used to conditionally terminate the execution of commands stored in a *batch buffer* initiated using a MI_BATCH_BUFFER_START command. **Programming Note**: This command is only valid with a 1st level batch buffer (bit 22 in MI_BATCH_BUFFER_START is set to '0')			
DWord	**Bit**	**Description**	
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode	
	28:23	**MI Command Opcode** Default Value: 36h MI_CONDITIONAL_BATCH_BUFFER_END Format: OpCode	

MI_CONDITIONAL_BATCH_BUFFER_END		
	22	**Use Global GTT** Project: All Default Value: 0h DefaultVaueDesc Format: U1 FormatDesc If set, this command will use the global GTT to translate the **Compare Address** and this command must be executing from a privileged (secure) batch buffer. If clear, the PPGTT will be used to translate the **Compare Address**. This bit will be ignored (and treated as if clear) if this command is executed from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer or directly from a ring buffer.
	21	**Compare Semaphore** Project: All Default Value: 0h DefaultVaueDesc Format: U1 FormatDesc If set, the value from the **Compare Data Dword** is compared to the value from the **Compare Address** in memory. If the value at **Compare Address is greater than the Compare Data Dword**, execution of current command buffer should continue. If clear, no comparison takes place.
	20	**Reserved**
	19:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:0	**Compare Data Dword** Data dword to compare memory. The Data dword is supplied by software to control execution of the command buffer. If the compare is enabled and the data at **Semaphore Address** is greater than this dword, the execution of the command buffer should continue.
2	31:3	**Compare Address** Qword address to fetch compare Mask (DW0) and Data Dword(DW1) from memory. HW will do AND operation on Mask(DW0) with Data Dword(DW1) and then compare the result against Semaphore Data Dword
	2:0	**Reserved** Project: All Format: MBZ

1.2.6 MI_BATCH_BUFFER_START

The MI_BATCH_BUFFER_START command format follows:

MI_BATCH_BUFFER_START

Project: All
Default Value: 00000000h
Engine: **Video**

The MI_BATCH_BUFFER_START command is used to initiate the execution of commands stored in a *batch buffer.* For restrictions on the location of batch buffers, see Batch Buffers in the Device Programming Interface chapter of *MI Functions*.

The batch buffer can be specified as secure or non-secure, determining the operations considered valid when initiated from within the buffer and any attached (chained) batch buffers. See Batch Buffer Protection in the Device Programming Interface chapter of *MI Functions*.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 31h MI_BATCH_BUFFER_START Format: OpCode
	22	**2nd Level Batch Buffer** Project: [DevSNB +] The command streamer contains 3 storage elements; 1 for the ring head address, 1 for the batch head address, and 1 for the 2nd level batch head address. When performing batch buffer chaining, hardware simply updates the head pointer of the 1st level batch address storage. There is no stack in hardware. When this bit is set, hardware uses the 2nd level batch head address storage element. Upon MI_BATCH_BUFFER_END, it will automatically return to the 1st (traditional) level batch buffer address. this allows hardware to mimic a simple 3 level stack. **Value** / **Name** / **Description** / **Project** 0h / 1st level batch / Place the batch buffer address in the 1st (traditional) level batch address storage element / [DevSNB+] 1h / 2nd level batch / Place the batch buffer address in the 2nd level batch address storage element / [DevSNB+] **Programming Notes** • A non-secure 2nd level batch buffer ***cannot*** be called from a non-secure 1st (traditional) level batch buffer. • 2nd level batch buffer chaining is ***not*** supported
	21:10	**Reserved** Project: rmat: 3Z
	9	**Reserved** Project: Format:

MI_BATCH_BUFFER_START		
	8	**Buffer Security Indicator** Project: All Format: U32 When this command is executed from within a batch buffer (i.e., is a “chained” batch buffer command), this field is IGNORED and the next buffer in the chain inherits the initial buffer’s security characteristics. [DevSNB+] If this bit is set, this batch buffer is non-secure and cannot execute privileged commands nor access privileged (GGTT) memory. It will be accessed via the PPGTT. If clear, this batch buffer is secure and will be accessed via the GGTT. Note that MI_STORE_DATA_IMM to non-privileged memory (via the PPGTT) *is* allowed in a non-secure batch buffer. Format = MI_BufferSecurityType 1 = MIBUFFER_NONSECURE 0 = MIBUFFER_SECURE (GGTT space) This field must be ‘0’ unless the **Per-Process GTT Enable** is ‘1’
	7:0	**DWord Length** (Excludes D-Word 0,1) = 0
1	31:2	**Buffer Start Address** Format: Graphics Virtual Address[31:2] FormatDesc **Programming Notes** • A batch buffer initiated with this command must end either with a MI_BATCH_BUFFER_END command or by chaining to another batch buffer with an MI_BATCH_BUFFER_START command. • The selection of PPGTT vs. GGTT for the batch buffer is determined by the **Buffer Security Indicator** (bit 8).
	1:0	**Reserved** Project: rmat: 3Z

1.2.7 MI_FLUSH_DW

MI_FLUSH_DW			
Project:	DevSNB+	**Length Bias:**	2
Engine:	**Video**		

The MI_FLUSH_DW command is used to perform an internal "flush" operation. The parser pauses on an internal flush until all drawing engines have completed any pending operations. In addition, this command can also be used to:

Flush any dirty data to memory.

Invalidate the TLB cache inside the hardware

Usage note: After this command is completed with a Store DWord enabled, CPU access to graphics memory will be coherent (assuming the Render Cache flush is not inhibited).

[DevSNB]: An MI_NOOP with NOP_ID bit set must be programmed after the last MI_FLUSH_DW before head = tail

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 26h MI_FLUSH_DW Format: OpCode
	22	**Protected memory Enable** Project: DevSNB+ Format: U1 After completion of the flush, the hardware will limit all access to the Protected Content Memory. Only command streamer initiated cacheable writes are allowed to non-PCM memory.
	21	**Store Data Index** Project: DevSNB+ Format: U1 This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is actually an index into the hardware status page. If this bit is set, this command will index into the per-process hardware status page if executed from within a non-secure batch buffer and if the **Per-Process Virtual Address Space** bit is set. Else the Global HW status page is used.
	20:19	**Reserved** Project: All Format: MBZ
	18	**TLB Invalidate** Project: DevSNB+ Format: U1 If ENABLED, all TLBs will be invalidated once the flush operation is complete. Note that if the flush TLB invalidation mode is clear, a TLB invalidate will occur irrespective of this bit setting. This bit is only valid when the Post-Sync Operation field is a value of 1h or 3h.

MI_FLUSH_DW

	17	**Synchronize GFDT surface** Project: DevSNB+ Format: U1 If enabled, at the end of the current flush the last level cache is cleared of all the cachelines which have been marked with the special GFDT flags. Store DW must be enabled
	16	**Reserved** Project: All Format: MBZ
	15:14	**Post-Sync Operation** Project: DevSNB+ BitFieldDesc **Value** **Name** **Description** **Project** 0h — No write occurs as a result of this instruction. This can be used to implement a "trap" operation, etc. — DevSNB+ 1h — Write the QWord containing Immediate Data Low, High DWs to the Destination Address — DevSNB+ 2h — **Reserved** — DevSNB+ 3h — Write the 32-bit TIMESTAMP register to the Destination Address with granularity of 640ns. Upper 32-bits are tied to '0' — DevSNB+ **Programming Notes** If executed in non-secure batch buffer, the address given will be in a PPGTT address space. If in a secure ring or batch, address given will be in GGTT space
	13:9	**Reserved** Project: All Format: MBZ
	8	**Notify Enable** Project: DevSNB+ Format: U1 If ENABLED, a Sync Completion Interrupt will be generated (if enabled by the MI Interrupt Control registers) once the sync operation is complete. See Interrupt Control Registers in *Memory Interface Registers* for details.
	7	**Video Pipeline Cache invalidate** Project: DevSNB+ Format: U1 Enable the invalidation of the video cache at the end of this flush
	6	**Reserved** Project: DevSNB+ Format:

MI_FLUSH_DW

	5:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) = 2 for DWord, 3 for QWord Format: =n Total Length - 2 Project: All
1	31:3	**Address** Project: DevSNB+ Address: GraphicsAddress[31:3] Surface Type: U32 This field specifies Bits 31:3 of the Address where the DWord or QWord will be stored. Note that the address can only be QWord aligned, irrespective of data size.
	2	**Destination Address Type** Project: All Defines address space of Destination Address **Value** / **Name** / **Description** / **Project** 0h / PPGTT / Use PPGTT address space for DW write / All 1h / GGTT / Use GGTT address space for DW write / All **Programming Notes** Ignored if “No write” is the selected in Operation.
	1:0	**Reserved** Project: All Format: MBZ
2..3	31:0	**Immediate Data** Format: U32 Address: GraphicsAddress[31:0] Range 0..2^32-1 This field specifies the DWord value to be written to the targeted location. DW2 is the lower DW if QW is desired. Only valid when 15:14 in header is set to 1h [DevSNB A] To avoid hitting a known hardware bug, drivers cannot send a QW write when bit 5 of the address is ‘1’

1.2.8 MI_LOAD_REGISTER_IMM

The MI_LOAD_REGISTER_IMM command format is:

MI_LOAD_REGISTER_IMM			
Project:	All	**Length Bias:**	2
Engine:	**Video**		

The MI_LOAD_REGISTER_IMM command requests a write of up to a DWord constant supplied in the command to the specified Register Offset (i.e., offset into Memory-Mapped Register Range). The register is loaded before the next command is executed.

[DevSNB] The behavior of this command is controlled by Dword 3, Bit 8 (**Disable Register Access**) of the RINGBUF register. If this command is disallowed then the command stream converts it to a NOOP.

If this command is executed from a batch buffer then the behavior of this command is controlled by Dword 0, Bit 8 (**Security Indicator**) of the BATCH_BUFFER_START Command. If the batch buffer is non-secure then the command stream converts this command to a NOOP.

The following addresses should NOT be used for LRIs

```
1. 0x8800 - 0x88FF
2. >= 0xC0000
```

Limited LRI cycles to the Display Engine 0x40000-0xBFFFF) are allowed, but must be spaced to allow only one pending at a time. This can be done by issuing an SRM to the same address immediately after each LRI.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 22h MI_LOAD_REGISTER_IMM Format: OpCode
	22:12	**Reserved** Project: All Format: MBZ
	11:8	**Byte Write Disables** Project: All Format: Enable[4] (bit 8 corresponds to Data DWord [7:0]). Range: Must specify a valid register write operation. [11:8] is '1111', then the register write will not occur. [11:8] is '0000', then the register DW will be updated. Any other value, the behavior will be specifically specified by the register or the behavior is undefined.
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All

MI_LOAD_REGISTER_IMM		
1	31:0	**Reserved** Project: All Format: MBZ
	22:2	**Register Offset** Project: All Format: U30 This field specifies bits [22:2] of the offset into the Memory Mapped Register Range (i.e., this field specifies a DWord offset). Mapped
	1:0	**Reserved** Project: All Format: MBZ
2	31:0	**Data DWord** Project: All Format: U32 FormatDesc This field specifies the DWord value to be written to the targeted location.

1.2.9 MI_NOOP

The MI_NOOP command format is:

MI_NOOP			
Project:	All	**Length Bias:**	1
Engine:	**Video**		
The MI_NOOP command basically performs a “no operation” in the command stream and is typically used to pad the command stream (e.g., in order to pad out a batch buffer to a QWord boundary). However, there is one minor (optional) function this command can perform – a 22-bit value can be loaded into the MI NOPID register. This provides a general-purpose command stream tagging ("breadcrumb") mechanism (e.g., to provide sequencing information for a subsequent breakpoint interrupt).			

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 00h MI_NOOP Format: OpCode
	22:0	**Identification Number** Format: Enable FormatDesc: 1 = Write the NOP_ID register. 0 = Do not write the NOP_ID register. **Identification Number Register Write Enable:** This field enables the value in the Identification Number field to be written into the MI NOPID register. If disabled, that register is unmodified – making this command an effective “no operation” function.

MI_NOOP		
	21:0	**Identification Number** Project: All Format: U22 This field contains a 22-bit number which can be written to the MI NOPID register.

1.2.10 MI_REPORT_HEAD

The format of the MI_REPORT_HEAD command is:

MI_REPORT_HEAD			
Project:	All	**Length Bias:**	1
Engine:	**Video**		

The MI_REPORT_HEAD command causes the Head Pointer value of the ring buffer to be written to a cacheable (snooped) system memory location.

When the **Per-Process Virtual Address Space** bit is reset, the location written is relative to the address programmed in the Hardware Status Page Address Register.

Programming Notes:

- This command must not be executed from a Batch Buffer (Refer to the description of the HWS_PGA register).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 07h MI_REPORT_HEAD Format: OpCode
	22:0	**Reserved** Project: All Format: MBZ

1.2.11 MI_SEMAPHORE_MBOX

MI_SEMAPHORE_MBOX			
Project:	DevSNB+	**Length Bias:**	2
Engine:	**Video**		

This command is provided as alternative to MI_SEMAPHORE to provide mailbox-type semaphores where there is no update of the semaphore by the checking process (the consumer). Single-bit compare-and-update semantics are also provided. In either case, atomic access of semaphores need not be guaranteed by hardware as with the previous command. This command should eventually supersede the previous command.

Synchronization between contexts (especially between contexts running on 2 different engines) is provided by the MI_SEMAPHORE_MBOX command. Note that contexts attempting to synchronize in this fashion must be able to access a common memory location. This means the contexts must share the same virtual address space (have the same page directory), must have a common physical page mapped into both of their respective address spaces or the semaphore commands must be executing from a secure batch buffer or directly from a ring with the **Use Global GTT** bit set such that they are "privileged" and will use the (always shared) global GTT.

MI_SEMAPHORE with the **Update Semaphore** bit set (and the **Compare Semaphore** bit clear) implements the *Signal* command, while the *Wait* command is indicated by **Compare Semaphore** being set. Note that *Wait* can cause a context switch. *Signal* increments unconditionally.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 16h MI_SEMAPHORE_MBOX Format: OpCode
	22	**Use Global GTT** Project: All Format: U32 If set, this command will use the global GTT to translate the **Semaphore Address** and this command must be executing from a privileged (secure) batch buffer. If clear, the PPGTT will be used to translate the **Semaphore Address**. This bit will be ignored (and treated as if clear) if this command is executed from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer or directly from a ring buffer.
	21	**Update Semaphore** Project: All Format: U32 If set, the value from the **Semaphore Data Dword** is written to memory. If **Compare Semaphore** is also set, the semaphore is not updated if the semaphore comparison fails. If clear, the data at **Semaphore Address** is not changed.
	20	**Compare Semaphore** Project: All Format: U32 If set, the value from the **Semaphore Data Dword** is compared to the value from the **Semaphore Address** in memory. If the value at **Semaphore Address is greater than the Semaphore Data Dword**, execution is continued from the current command buffer. If clear, no comparison takes place. **Update Semaphore** *must* be set in this case.

MI_SEMAPHORE_MBOX		
	19	**Reserved** Project: All Format: MBZ
	18	**Compare Register** Project: DevSNB+ Format: Compare Type If set, data in MMIO register will be used for compare. If clear, data in memory will be used for compare.
	17:16	**Register Select** Project: DevSNB+ Format: Register Select If compare register is set in bit[18], this filed indicate which register will be used. 0: BCS register (VBSYNC) 1: [Reserved] 2: CS regiser (VRSYNC) 3: Reserved
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:0	**Semaphore Data Dword** Project: All Format: U32 Data dword to compare/update memory. The Data dword is supplied by software to control execution of the command buffer. If the compare is enabled and the data at **Semaphore Address** is greater than this dword, the execution of the command buffer continues.
2	31:2	**PointerBitFieldName/MMIO Register Address** Project: All Address: GraphicsVirtualAddress[31:2] Surface Type: Semaphore if Compare Register bit[18] is cleared, this field if the Graphics Memory Address of the 32 bit value for the semaphore. If Compare Register bit[18] is set, this field is the MMIO address of the register for the semaphore.
	1:0	**Reserved** Project: All Format: MBZ

1.2.12 MI_STORE_REGISTER_MEM

MI_STORE_REGISTER_MEM

Project:	DevSNB+	**Length Bias:**	2
Engine:	**Video**		

The MI_STORE_REGISTER_MEM command requests a register read from a specified memory mapped register location in the device and store of that DWord to memory. The register address is specified along with the command to perform the read.

Programming Notes:

- The command temporarily halts command execution.
- The memory address for the write is snooped on the host bus.
- This command will cause undefined data to be written to memory if given register addresses for the PGTBL_CTL_0 or FENCE registers

The following addresses should NOT be used for SRMs

1. 0x8800 - 0x88FF
2. >= 0x40000

The only exception is an SRM cycle to 0x40000-0xBFFFF when used as part of the LRI read-after-write requirement.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 24h MI_STORE_REGISTER_MEM Format: OpCode
	22	**Use Global GTT** Project: All This bit *must* be '1' if the **Per Process GTT Enable** bit is clear. **Value** / **Name** / **Description** / **Project** 0h / Per Process Graphics Address / / All 1h / Global Graphics Address / This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer. / All **Programming Notes:** [DevSNB] This will be ignored and treated as if clear when executing from a PPGTT (i.e. runlist mode "non-secure") batch buffer
	21:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) Format: =n Total Length - 2

MI_STORE_REGISTER_MEM		
1	31:23	**Reserved** Project: DevSNB+ Format: MBZ
	22:2	**Register Address** Project: All Address: MMIO Address[22:2] Surface Type: MMIO Register This field specifies Bits 22:2 of the Register offset the DWord will be read from. As the register address must be DWord-aligned, Bits 1:0 of that address MBZ. **Programming Notes** — **Project** Storing a VGA register is not permitted and will store an UNDEFINED value. — All The values of PGTBL_CTL0 or any of the FENCE registers cannot be stored to memory; UNDEFINED values will be written to memory if the addresses of these registers are specified. — All
	1:0	**Reserved** Project: All Format: MBZ
2	31:2	**Memory Address** Project: DevSNB+ Address: GraphicsAddress[31:2] Surface Type: MMIO Register This field specifies the address of the memory location where the register value specified in the DWord above will be written. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[31:2] for a DWord register
	1:0	**Reserved** Project: All Format: MBZ

1.2.13 MI_STORE_DATA_IMM

The MI_STORE_DATA_IMM command format is:

MI_STORE_DATA_IMM			
Project:	All	**Length Bias:**	2
Engine:	**Video**		

The MI_STORE_DATA_IMM command requests a write of the QWord or DWord constant supplied in the packet to the specified Memory Address. As the write targets a System Memory Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes:

This command should not be used within a "non-secure" batch buffer to access global virtual space. Doing so will cause the command parser to perform the write with byte enables turned off. This command can be used within ring buffers and/or "secure" batch buffers.

[DevSNB] **Use Global GTT** will be ignored and treated as if clear when executing from a PPGTT (i.e. runlist mode "non-secure") batch buffer

This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll un-cached memory or device registers).

This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete "eventually", there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 20h MI_STORE_DATA_IMM Format: OpCode
	22	**Use Global GTT** Project: All Format: U32 If set, this command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer. If clear, the PPGTT will be used. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit *must* be '1' if the **Per Process GTT Enable** bit is clear. **Programming Note:** [DevSNB] This bit will be ignored and treated as if clear when executing from a non-privileged batch buffer
	21:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: Excludes DWord (0,1) = 3 for QWord, 2 for DWord Format: Total Length - 2

MI_STORE_DATA_IMM		
1	31:0	**Reserved** Project: All Format: MBZ
2	31:2	**Address** Format: Bits[31:2] of a Graphics Virtual Address This field specifies Bits 31:2 of the Address where the DWord will be stored. As the store address must be DWord-aligned, Bits 1:0 of that address MBZ. This address must be 8B aligned for a store “QW” command.
	1:0	**Reserved** Project: All Format: MBZ
3	31:0	**Data DWord 0** Format: U32 FormatDesc This field specifies the DWord value to be written to the targeted location. For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).
4	31:0	**Data DWord 1** Format: U32 FormatDesc This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

1.2.14 MI_STORE_DATA_INDEX

The MI_STORE_DATA_INDEX command format is:

MI_STORE_DATA_INDEX			
Project:	All	**Length Bias:**	2
Engine:	**Video**		

The MI_STORE_DATA_INDEX command requests a write of the data constant supplied in the packet to the specified offset from the System Address defined by the Hardware Status Page Address Register. As the write targets a System Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes:

- Use of this command with an invalid or uninitialized value in the Hardware Status Page Address Register is UNDEFINED.
- This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll uncached memory or device registers).
- This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete "eventually", there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 21h MI_STORE_DATA_INDEX Format: OpCode
	22	**Reserved** Project: All Format: MBZ
	21	**Use Per-Process Hardware Status Page** Project: All Format: If this bit is set, this command will index into the per-process hardware status page at offset 20K from the LRCA. If clear, the Global Hardware Status Page will be indexed. All other devices: Reserved: MBZ. **Programming Notes:** [DevSNB] This will be ignored and treated as if set when executing from a PPGTT batch buffer
	20:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) = 2 for QWord Format: =n Total Length - 2 Project: All

MI_STORE_DATA_INDEX		
1	31:12	**Reserved** Project: Format: MBZ
	11:2	**Offset** Project: All Format: U10 FormatDesc; zero-based DWord offset into the HW status page Address: GraphicsAddress[31:0] Surface Type: U32 Range [16, 1023]. This field specifies the offset (into the hardware status page) to which the data will be written. Note that the first few DWords of this status page are reserved for special-purpose data storage – targeting these reserved locations via this command is UNDEFINED. For a QWord write, the offset is valid down to bit 3 only.
	1:0	**Reserved** Project: Format: MBZ
2	31:0	**Data DWord 0** Format: U32 FormatDesc This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).
3	31:0	**Data Word 1** Format: U32 FormatDesc This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

1.2.15 MI_SUSPEND_FLUSH

MI_SUSPEND_FLUSH			
Project:	All	**Length Bias:**	1
Engine:	**Video**		
Blocks MMIO sync flush or any flushes related to VT-d while enabled.			

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Bh MI_SUSPEND_FLUSH Format: OpCode
	22:1	**Reserved** Project: All Format: MBZ
	0	**Suspend Flush** Project: All Default Value: 0h DefaultVaueDesc Format: Enable FormatDesc This field suspends flush due to sync flush or implicit flush generated during VTD enable, disable and IOTLB invalidation. **Value** **Name** **Description** **Project** 0h Disable All 1h Enable All

1.2.16 MI_USER_INTERRUPT

MI_USER_INTERRUPT			
Project:	All	**Length Bias:**	1
Engine:	**Video**		
The MI_USER_INTERRUPT command is used to generate a User Interrupt condition. The parser will continue parsing after processing this command.			

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 02h MI_USER_INTERRUPT Format: OpCode
	22:0	**Reserved** Project: All Format: MBZ

1.2.17 MI_UPDATE_GTT

1.2.17.1 MI_UPDATE_GTT [DevSNB]

MI_UPDATE_GTT

Project:	DevSNB	**Length Bias:**	2
Engine:	Video		

The MI_UPDATE_GTT command is used to update GTT page table entries in a coherent manner and at a predictable place in the command flow.

An MI_FLUSH should be placed before this command, since work associated with preceding commands that are still in the pipeline may be referencing GTT entries that will be changed by its execution. The flush will also invalidate TLBs and read caches that may become invalid as a result of the changed GTT entries. MI_FLUSH is not required if it can be guaranteed that the pipeline is free of any work that relies on changing GTT entries (such as MI_UPDATE_GTT contained in a paging DMA buffer that is doing only update/mapping activities and no rendering).

This is a privileged command. This command will be converted to a no-op and an error flagged if it is executed from within a non-secure batch buffer.

MI_UPDATE_GTT contents must be in address/data pair. This is different from the render CS definition. PPGTT updates cannot be done via MI_UPDATE_GTT, gfx driver will have to use storeDW for PPGTT inline updates.

Note that MI_UPDATE_GTT is mainly for the pages that are strictly used by PG. If driver chooses to update the CPU used pages thru MI_UPDATE_GTT, it needs to write to MMIO address x101008 (any value) to ensure system agent TLBs are invalidated before the new pages can be used.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 23h MI_UPDATE_GTT Format: OpCode
	22	**Use Global GTT** Project: All Reserved: Must be 1h. Updating Per Process Graphics Address is not supported **Value** / **Name** / **Description** / **Project** 0h / Per Process Graphics Address / Illegal, not supported. / All 1h / Global Graphics Address / This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer. / All

MI_UPDATE_GTT		
	21:6	**Reserved** Project: All Format: MBZ
	5:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) Format: =n Total Length – 2, max 61
1..n+1	63:44	**Entry Address** Project: All Address: GraphicsAddress[31:12] This field simply holds the DW offset of the first table entry to be modified. Note that one or more of the upper bits may need to be 0, i.e., for a 2G aperture, bit 31 MBZ.
	43:32	**Reserved** Project: All Format: MBZ
	31:0	**Entry Data** Project: All Format: Page Table Entry This Dword becomes the new page table entry. See PPGTT/Global GTT Table Entries (PTEs) in Memory Interface Registers.

1.2.18 MI_WAIT_FOR_EVENT

MI_WAIT_FOR_EVENT			
Project:	All	**Length Bias:**	1
Engine:	**Video**		

The MI_WAIT_FOR_EVENT command is used to pause command stream processing until a specific event occurs or while a specific condition exists. See Wait Events/Conditions, Device Programming Interface in *MI Functions*. Only one event/condition can be specified -- specifying multiple events is UNDEFINED.

The effect of the wait operation depends on the source of the command. If executed from a batch buffer, the parser will halt (and suspend command arbitration) until the event/condition occurs. If executed from a ring buffer, further processing of that ring will be suspended, although command arbitration (from other rings) will continue. Note that if a specified condition does not exist (the condition code is inactive) at the time the parser executes this command, the parser proceeds, treating this command as a no-operation.

If execution of this command from a primary ring buffer causes a wait to occur, the active ring buffer will *effectively* give up the remainder of its time slice (required in order to enable arbitration from other primary ring buffers).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode

MI_WAIT_FOR_EVENT

	28:23	**MI Command Opcode** Default Value: 03h MI_WAIT_FOR_EVENT Format: OpCode
	22:20	**Reserved** Project: All Format: MBZ
	19:16	**Condition Code Wait Select** Project: All This field enables a wait for the duration that the corresponding condition code is active. These enable select one of 15 condition codes in the EXCC register, that cause the parser to wait until that condition-code in the EXCC is cleared. **Value** / **Name** / **Description** / **Project** 0h / Not enabled / Condition Code Wait Not Enabled / All 1h-5h / Enable / Condition Code select enabled; selects one of 5 codes, 0 – 4 / All 6h-15h / Reserved / All **Programming Notes** Note that not all condition codes are implemented. The parser operation is UNDEFINED if an unimplemented condition code is selected by this field. The description of the EXCC register (*Memory Interface Registers*) lists the codes that are implemented.
	15:0	**Reserved** Project: All Format: MBZ

Revision History

Revision Number	Description	Revision Date
1.0	First 2011 OpenSource edition	May 2011

§§

Intel® OpenSource HD Graphics Programmer's Reference Manual (PRM)

Volume 1 Part 5: Graphics Core – Blitter Engine (SandyBridge)

For the 2011 Intel Core Processor Family

May 2011

Revision 1.0

NOTICE:

This document contains information on products in the design phase of development, and Intel reserves the right to add or remove product features at any time, with or without changes to this open source documentation.

Doc Ref #: IHD-OS-V1 Pt5 – 05 11

Creative Commons License

You are free to Share — to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

No Derivative Works. You may not alter, transform, or build upon this work.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The SandyBridge chipset family, Havendale/Auburndale chipset family, Intel® 965 Express Chipset Family, Intel® G35 Express Chipset, and Intel® 965GMx Chipset Mobile Family Graphics Controller may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel® sales office or your distributor to obtain the latest specifications and before placing your product order. I2C is a two-wire communications bus/protocol developed by Philips. SMBus is a subset of the I2C bus/protocol and was developed by Intel®. Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

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

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

Copyright © 2011, Intel Corporation. All rights reserved.

Contents

1. BLT Engine 5
1.1 Introduction 5
1.2 Classical BLT Engine Functional Description 5
1.2.1 Basic BLT Functional Considerations 7
1.2.2 Basic Graphics Data Considerations 18
1.2.3 BLT Programming Examples 23
1.3 BLT Instruction Overview 29
1.4 BLT Engine State 29
1.5 Cacheable Memory Support 30
1.6 Device Cache Coherency: Render & Texture Caches 31
1.7 BLT Engine Instructions 31
1.7.1 Blt Programming Restrictions 31
1.8 Fill/Move Instructions 32
1.8.1 COLOR_BLT (Fill) 32
1.8.2 SRC_COPY_BLT (Move) 32
1.9 2D (X,Y) BLT Instructions 33
1.9.1 XY_SETUP_BLT 35
1.9.2 XY_SETUP_MONO_PATTERN_SL_BLT 37
1.9.3 XY_SETUP_CLIP_ BLT 38
1.9.4 XY_PIXEL_BLT 38
1.9.5 XY_SCANLINES_BLT 39
1.9.6 XY_TEXT_BLT 40
1.9.7 XY_TEXT_IMMEDIATE_BLT 40
1.9.8 XY_COLOR_BLT 41
1.9.9 XY_PAT_BLT 42
1.9.10 XY_PAT_CHROMA_BLT 44
1.9.11 XY_PAT_BLT_IMMEDIATE 45
1.9.12 XY_PAT_CHROMA_BLT_IMMEDIATE 46
1.9.13 XY_MONO_PAT_BLT 48
1.9.14 XY_MONO_PAT_FIXED_BLT 49
1.9.15 XY_SRC_COPY_BLT 57
1.9.16 XY_SRC_COPY_CHROMA_BLT 58
1.9.17 XY_MONO_SRC_COPY_BLT 60
1.9.18 XY_MONO_SRC_COPY_ IMMEDIATE_BLT 61
1.9.19 XY_FULL_BLT 62
1.9.20 XY_FULL_IMMEDIATE_PATTERN_BLT 64
1.9.21 XY_FULL_MONO_SRC_BLT 66
1.9.22 XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT 67
1.9.23 XY_FULL_MONO_PATTERN_BLT 69
1.9.24 XY_FULL_MONO_PATTERN_MONO_SRC_BLT 71
1.10 BLT Engine Instruction Field Definitions 73
1.10.1 BR00—BLT Opcode & Control 73
1.10.2 BR01—Setup BLT Raster OP, Control, and Destination Offset 76
1.10.3 BR05—Setup Expansion Background Color 78
1.10.4 BR06—Setup Expansion Foreground Color 79
1.10.5 BR07—Setup Color Pattern Address 79
1.10.6 BR09—Destination Address 80
1.10.7 BR11—BLT Source Pitch (Offset) 81
1.10.8 BR12—Source Address 82
1.10.9 BR13—BLT Raster OP, Control, and Destination Pitch 82

1.10.10 BR14—Destination Width and Height 84
1.10.11 BR15—Color Pattern Address 85
1.10.12 BR16—Pattern Expansion Background and Solid Pattern Color 86
1.10.13 BR17—Pattern Expansion Foreground Color 86
1.10.14 BR18—Source Expansion Background, and Destination Color 87
1.10.15 BR19—Source Expansion Foreground Color 87
2. Blitter (Blt) Engine Command Streamer 88
2.1 Registers for Blitter Engine 88
2.1.1 Introduction 88
2.1.2 GAB PWR CTX STORAGE REGISTERS 89
2.1.3 Virtual Memory Control 90
2.1.4 Mode and Misc Ctrl Registers 93
2.1.5 BCS_RINGBUF—Ring Buffer Registers 101
2.1.6 Interrupt Control Registers 106
2.1.7 Logical Context Support 113
2.1.8 Software Control Bit Definitions 114
2.2 Memory Interface Commands for Blitter Engine 114
2.2.1 Introduction 114
2.2.2 MI_ARB_CHECK 115
2.2.3 MI_BATCH_BUFFER_END 115
2.2.4 MI_BATCH_BUFFER_START 116
2.2.5 MI_FLUSH_DW 119
2.2.6 MI_LOAD_REGISTER_IMM 122
2.2.7 MI_NOOP 123
2.2.8 MI_REPORT_HEAD 124
2.2.9 MI_SEMAPHORE_MBOX 124
2.2.10 MI_STORE_REGISTER_MEM 126
2.2.11 MI_STORE_DATA_IMM 128
2.2.12 MI_STORE_DATA_INDEX 129
2.2.13 MI_SUSPEND_FLUSH 131
2.2.14 MI_UPDATE_GTT 132
2.2.15 MI_USER_INTERRUPT 134
2.2.16 MI_WAIT_FOR_EVENT [DevSNB] 134
2.2.17 MSG_LOAD_SL – Load Scan Lines Message 135

1. BLT Engine

1.1 Introduction

2D Rendering can be divided into 2 categories: classical BLTs, described here, and 3D BLTs.

3D BLTs are operations which can take advantage of the 3D drawing engine's functionality and access patterns. Functions such as Alpha BLTs, arithmetic (bilinear) stretch BLTs, rotations, transposing pixel maps, color space conversion, and DIBs are all considered 3D BLTs and are covered in the 3D rendering section. DIBs can be thought of as an indexed texture which uses the texture palette for performing the data translation. All drawing engines have swappable context. The same hardware can be used by multiple driver threads where the current state of the hardware is saved to memory and the appropriate state is loaded from memory on thread switches.

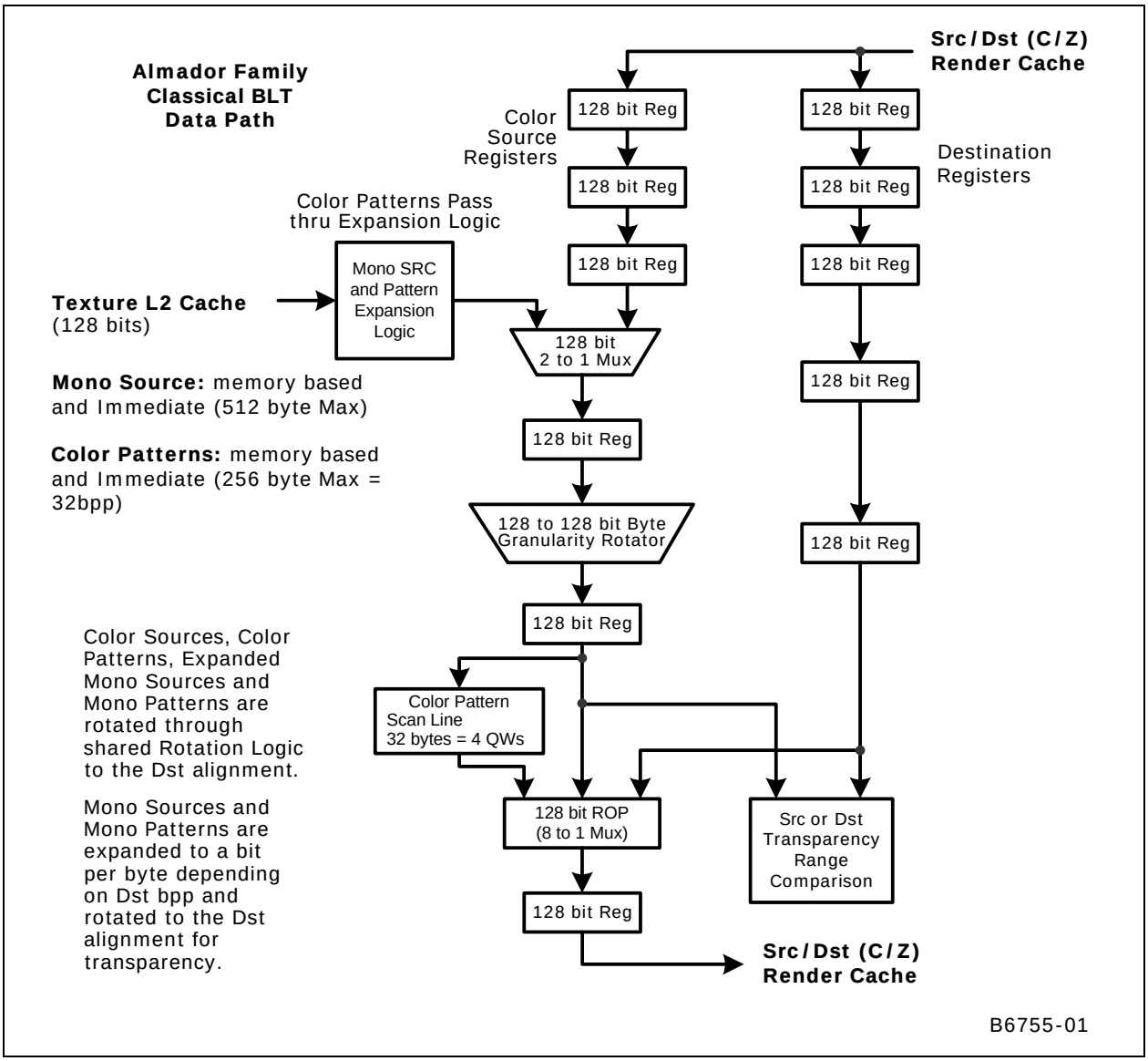
All operands for both 3D and classical BLTs can be in graphics aperture or cacheable system memory. Some operands can be immediates which are sent through the command stream. Immediate operands are: patterns, monochrome sources, DIB palettes, and DIB source operands. All non-monochrome operands which are not tiled have a stride granularity of a double-word (4 bytes).

The classical BLT commands support both linear addressing and X, Y coordinates with and without clipping. All X1 and Y1 destination and clipping coordinates are inclusive, while X2 and Y2 are exclusive. Currently, only destination coordinates can be negative. The source and clipping coordinates must be positive. If clipping is disabled, but a negative destination coordinate is specified, the negative coordinate is clipped to 0. Linear address BLT commands must supply a non-zero height and width. If either height or width = 0, then no accesses occur.

1.2 Classical BLT Engine Functional Description

The graphics controller provides a hardware-based BLT engine to off load the work of moving blocks of graphics data from the host CPU. Although the BLT engine is often used simply to copy a block of graphics data from the source to the destination, it also has the ability to perform more complex functions. The BLT engine is capable of receiving three different blocks of graphics data as input as shown in the figure below. The source data may exist in the frame buffer or the Graphics aperture. The pattern data always represents an 8x8 block of pixels that can be located in the frame buffer, Graphics aperture, or passed through a command packet. The pattern data must be located in linear memory. The data already residing at the destination may also be used as an input. The destination data can also be located in the frame buffer or graphics aperture.

Figure 1-1. Block Diagram and Data Paths of the BLT Engine



The BLT engine may use any combination of these three different blocks of graphics data as operands, in both bit-wise logical operations to generate the actual data to be written to the destination, and in per-pixel write-masking to control the writing of data to the destination. It is intended that the BLT engine will perform these bit-wise and per-pixel operations on color graphics data that is at the same color depth that the rest of the graphics system has been set. However, if either the source or pattern data is monochrome, the BLT engine has the ability to put either block of graphics data through a process called “color expansion” that converts monochrome graphics data to color. Since the destination is often a location in the on-screen portion of the frame buffer, it is assumed that any data already at the destination will be of the appropriate color depth.

1.2.1 Basic BLT Functional Considerations

1.2.1.1 Color Depth Configuration and Color Expansion

The graphics system and BLT engine can be configured for color depths of 8, 16, and 32 bits per pixel.

The configuration of the BLT engine for a given color depth dictates the number of bytes of graphics data that the BLT engine will read and write for each pixel while performing a BLT operation. It is assumed that any graphics data already residing at the destination which is used as an input is already at the color depth to which the BLT engine is configured. Similarly, it is assumed that any source or pattern data used as an input has this same color depth, unless one or both is monochrome. If either the source or pattern data is monochrome, the BLT engine performs a process called “color expansion” to convert such monochrome data to color at the color depth to which the BLT engine has been set.

During “color expansion” the individual bits of monochrome source or pattern data that correspond to individual pixels are converted into 1, 2, or 4 bytes (which ever is appropriate for the color depth to which the BLT engine has been set). If a given bit of monochrome source or pattern data carries a value of 1, then the byte(s) of color data resulting from the conversion process are set to carry the value of a specified foreground color. If a given bit of monochrome source or pattern data carries a value of 0, the resulting byte(s) are set to the value of a specified background color or not written if transparency is selected.

The BLT engine is set to a default configuration color depth of 8, 16, or 32 bits per pixel through BLT command packets. Whether the source and pattern data are color or monochrome must be specified using command packets. Foreground and background colors for the color expansion of both monochrome source and pattern data are also specified through the command packets. The source foreground and background colors used in the color expansion of monochrome source data are specified independently of those used for the color expansion of monochrome pattern data.

1.2.1.2 Graphics Data Size Limitations

The BLT engine is capable of transferring very large quantities of graphics data. Any graphics data read from and written to the destination is permitted to represent a number of pixels that occupies up to 65,536 scan lines and up to 32,768 bytes per scan line at the destination. The maximum number of pixels that may be represented per scan line’s worth of graphics data depends on the color depth.

Any source data used as an input must represent the same number of pixels as is represented by any data read from or written to the destination, and it must be organized so as to occupy the same number of scan lines and pixels per scan line.

The actual number of scan lines and bytes per scan line required to accommodate data read from or written to the destination are set in the destination width & height registers or using X and Y coordinates within the command packets. These two values are essential in the programming of the BLT engine, because the engine uses these two values to determine when a given BLT operation has been completed.

1.2.1.3 Bit-Wise Operations

The BLT engine can perform any one of 256 possible bit-wise operations using various combinations of the three previously described blocks of graphics data that the BLT engine can receive as input. These

256 possible bit-wise operations are designed to be compatible with the manner in which raster operations are specified in the standard BLT parameter block, without translation.

The choice of bit-wise operation selects which of the three inputs will be used, as well as the particular logical operation to be performed on corresponding bits from each of the selected inputs. The BLT engine automatically foregoes reading any form of graphics data that has not been specified as an input by the choice of bit-wise operation. An 8-bit code written to the raster operation field of the command packets chooses the bit-wise operation. The following table lists the available bit-wise operations and their corresponding 8-bit codes.

Table 1-1. Bit-Wise Operations and 8-Bit Codes (00-3F)

Code	Value Written to Bits at Destination
00	writes all 0's
01	not(D or (P or S)))
02	D and (not(P or S))
03	not(P or S)
04	S and (not(D or P))
05	not(D or P)
06	not(P or (not(D xor S)))
07	not(P or (D and S))
08	S and (D and (notP))
09	not(P or (D xor S))
0A	D and (notP)
0B	not(P or (S and (notD)))
0C	S and (notP)
0D	not(P or (D and (notS)))
0E	not(P or (not(D or S)))
0F	notP
10	P and (not(D or S))
11	not(D or S)
12	not(S or (not(D xor P)))
13	not(S or (D and P))
14	not(D or (not(P xor S)))
15	not(D or (P and S))
16	P xor (S xor (D and (not(P and S))))
17	not(S xor ((S xor P) and (D xor S)))
18	(S xor P) and (P xor D)
19	not(S xor (D and (not(P and S))))
1A	P xor (D or (S and P))
1B	not(S xor (D and (P xor S)))
1C	P xor (S or (D and P))
1D	not(D xor (S and (P xor D)))
1E	P xor (D or S)
1F	not(P and (D or S))

Code	Value Written to Bits at Destination
20	D and (P and (notS))
21	not(S or(D xor P))
22	D and (notS)
23	not(S or (P and (notD)))
24	(S xor P) and (D xor S)
25	not(P xor (D and (not(S and P))))
26	S xor (D or (P and S))
27	S xor (D or (not(P xor S)))
28	D and (P xor S)
29	not(P xor (S xor (D or (P and S))))
2A	D and (not(P and S))
2B	not(S xor ((S xor P) and (P xor D)))
2C	S xor (P and (D or S))
2D	P xor (S or (notD))
2E	P xor (S or (D xor P))
2F	not(P and (S or (notD)))
30	P and (notS)
31	not(S or (D and (notP)))
32	S xor (D or (P or S))
33	notS
34	S xor (P or (D and S))
35	S xor (P or (not(D xor S)))
36	S xor (D or P)
37	not(S and (D or P))
38	P xor (S and (D or P))
39	S xor (P or (notD))
3A	S xor (P or (D xor S))
3B	not(S and (P or (notD)))
3C	P xor S
3D	S xor (P or (not(D or S)))
3E	S xor (P or (D and (notS)))
3F	not(P and S)

Notes: S = Source Data
P = Pattern Data
D = Data Already Existing at the Destination

Table 1-2. Bit-Wise Operations and 8-bit Codes (40 - 7F)

Code	Value Written to Bits at Destination
40	P and (S and (notD))
41	not(D or (P xor S))
42	(S xor D) and (P xor D)
43	not(S xor (P and (not(D and S))))
44	S and (notD)
45	not(D or (P and (notS)))
46	D xor (S or (P and D))
47	not(P xor (S and (D xor P)))
48	S and (D xor P)
49	not(P xor (D xor (S or (P and D))))
4A	D xor (P and (S or D))
4B	P xor (D or (notS))
4C	S and (not(D and P))
4D	not(S xor ((S xor P) or (D xor S)))
4E	P xor (D or (S xor P))
4F	not(P and (D or (notS)))
50	P and (notD)
51	not(D or (S and (notP)))
52	D xor (P or (S and D))
53	not(S xor (P and (D xor S)))
54	not(D or (not(P or S)))
55	notD
56	D xor (P or S)
57	not(D and (P or S))
58	P xor (D and (S or P))
59	D xor (P or (notS))
5A	D xor P
5B	D xor (P or (not(S or D)))
5C	D xor (P or (S xor D))
5D	not(D and (P or (notS)))
5E	D xor (P or (S and (notD)))
5F	not(D and P)

Code	Value Written to Bits at Destination
60	P and (D xor S)
61	not(D xor (S xor (P or (D and S))))
62	D xor (S and (P or D))
63	S xor (D or (notP))
64	S xor (D and (P or S))
65	D xor (S or (notP))
66	D xor S
67	S xor (D or (not(P or S)))
68	not(D xor (S xor (P or (not(D or S)))))
69	not(P xor (D xor S))
6A	D xor (P and S)
6B	not(P xor (S xor (D and (P or S))))
6C	S xor (D and P)
6D	not(P xor (D xor (S and (P or D))))
6E	S xor (D and (P or (notS)))
6F	not(P and (not(D xor S)))
70	P and (not(D and S))
71	not(S xor ((S xor D) and (P xor D)))
72	S xor (D or (P xor S))
73	not(S and (D or (notP)))
74	D xor (S or (P xor D))
75	not(D and (S or (notP)))
76	S xor (D or (P and (notS)))
77	not(D and S)
78	P xor (D and S)
79	not(D xor (S xor (P and (D or S))))
7A	D xor (P and (S or (notD)))
7B	not(S and (not(D xor P)))
7C	S xor (P and (D or (notS)))
7D	not(D and (not(P xor S)))
7E	(S xor P) or (D xor S)
7F	not(D and (P and S))

Notes: S = Source Data
P = Pattern Data
D = Data Already Existing at the Destination

Table 1-3. Bit-Wise Operations and 8-bit Codes (80 - BF)

Code	Value Written to Bits at Destination
80	D and (P and S)
81	not((S xor P) or (D xor S))
82	D and (not(P xor S))
83	not(S xor (P and (D or (notS))))
84	S and (not(D xor P))
85	not(P xor (D and (S or (notP))))
86	D xor (S xor (P and (D or S)))
87	not(P xor (D and S))
88	D and S
89	not(S xor (D or (P and (notS))))
8A	D and (S or (notP))
8B	not(D xor (S or (P xor D)))
8C	S and (D or (notP))
8D	not(S xor (D or (P xor S)))
8E	S xor ((S xor D) and (P xor D))
8F	not(P and (not(D and S)))
90	P and (not(D xor S))
91	not(S xor (D and (P or (notS))))
92	D xor (P xor (S and (D or P)))
93	not(S xor (P and D))
94	P xor (S xor (D and (P or S)))
95	not(D xor (P and S))
96	D xor (P xor S)
97	P xor (S xor (D or (not(P or S))))
98	not(S xor (D or (not(P or S))))
99	not(D xor S)
9A	D xor (P and (notS))
9B	not(S xor (D and (P or S)))
9C	S xor (P and (notD))
9D	not(D xor (S and (P or D)))
9E	D xor (S xor (P or (D and S)))
9F	not(P and (D xor S))

Code	Value Written to Bits at Destination
A0	D and P
A1	not(P xor (D or (S and (notP))))
A2	D and (P or (notS))
A3	not(D xor (P or (S xor D)))
A4	not(P xor (D or (not(S or P))))
A5	not(P xor D)
A6	D xor (S and (notP))
A7	not(P xor (D and (S or P)))
A8	D and (P or S)
A9	not(D xor (P or S))
AA	D
AB	D or (not(P or S))
AC	S xor (P and (D xor S))
AD	not(D xor (P or (S and D)))
AE	D or (S and (notP))
AF	D or (notP)
B0	P and (D or (notS))
B1	not(P xor (D or (S xor P)))
B2	S xor ((S xor P) or (D xor S))
B3	not(S and (not(D and P)))
B4	P xor (S and (notD))
B5	not(D xor (P and (S or D)))
B6	D xor (P xor (S or (D and P)))
B7	not(S and (D xor P))
B8	P xor (S and (D xor P))
B9	not(D xor (S or (P and D)))
BA	D or (P and (notS))
BB	D or (notS)
BC	S xor (P and (not(D and S)))
BD	not((S xor D) and (P xor D))
BE	D or (P xor S)
BF	D or (not(P and S))

Notes: S = Source Data
P = Pattern Data
D = Data Already Existing at the Destination

Table 1-4. Bit-Wise Operations and 8-bit Codes (C0 - FF)

Code	Value Written to Bits at Destination
C0	P and S
C1	not(S xor (P or (D and (notS))))
C2	not(S xor (P or (not(D or S))))
C3	not(P xor S)
C4	S and (P or (notD))
C5	not(S xor (P or (D xor S)))
C6	S xor (D and (notP))
C7	not(P xor (S and (D or P)))
C8	S and (D or P)
C9	not(S xor (P or D))
CA	D xor (P and (S xor D))
CB	not(S xor (P or (D and S)))
CC	S
CD	S or (not(D or P))
CE	S or (D and (notP))
CF	S or (notP)
D0	P and (S or (notD))
D1	not(P xor (S or (D xor P)))
D2	P xor (D and (notS))
D3	not(S xor (P and (D or S)))
D4	S xor ((S xor P) and (P xor D))
D5	not(D and (not(P and S)))
D6	P xor (S xor (D or (P and S)))
D7	not(D and (P xor S))
D8	P xor (D and (S xor P))
D9	not(S xor (D or (P and S)))
DA	D xor (P and (not(S and D)))
DB	not((S xor P) and (D xor S))
DC	S or (P and (notD))
DD	S or (notD)
DE	S or (D xor P)
DF	S or (not(D and P))

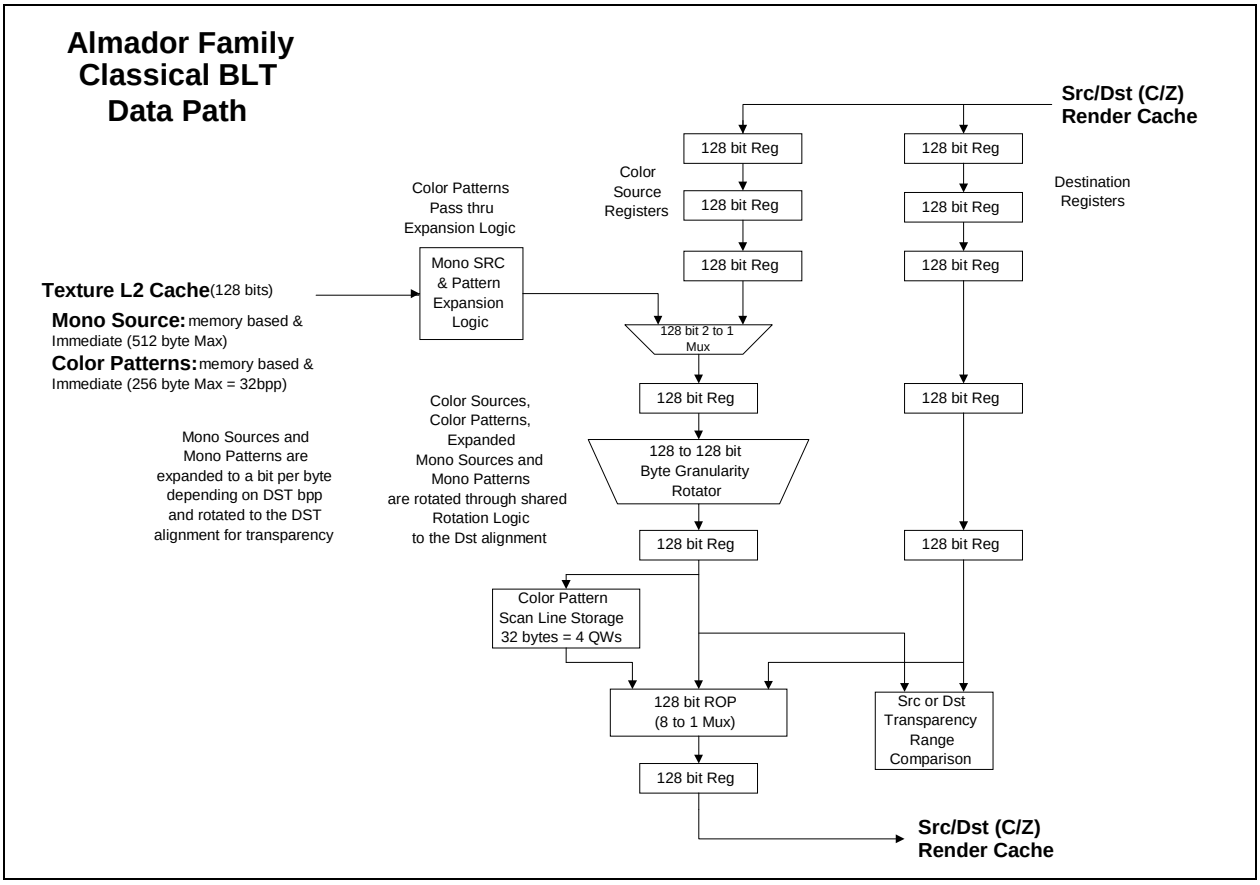
Code	Value Written to Bits at Destination
E0	P and (D or S)
E1	not(P xor (D or S))
E2	D xor (S and (P xor D))
E3	not(P xor (S or (D and P)))
E4	S xor (D and (P xor S))
E5	not(P xor (D or (S and P)))
E6	S xor (D and (not(P and S)))
E7	not((S xor P) and (P xor D))
E8	S xor ((S xor P) and (D xor S))
E9	not(D xor (S xor (P and (not(D and S)))))
EA	D or (P and S)
EB	D or (not(P xor S))
EC	S or (D and P)
ED	S or (not(D xor P))
EE	D or S
EF	S or (D or (notP))
F0	P
F1	P or (not(D or S))
F2	P or (D and (notS))
F3	P or (notS)
F4	P or (S and (notD))
F5	P or (notD)
F6	P or (D xor S)
F7	P or (not(D and S))
F8	P or (D and S)
F9	P or (not(D xor S))
FA	D or P
FB	D or (P or (notS))
FC	P or S
FD	P or (S or (notD))
FE	D or (P or S)
FF	writes all 1's

Notes: S = Source Data
P = Pattern Data
D = Data Already Existing at the Destination

1.2.1.4 Per-Pixel Write-Masking Operations

The BLT engine is able to perform per-pixel write-masking with various data sources used as pixel masks to constrain which pixels at the destination are to be written to by the BLT engine. As shown in the figure below, either monochrome source or monochrome pattern data may be used as pixel masks. Color pattern data cannot be used. Another available pixel mask is derived by comparing a particular color range per color channel to either the color already specified for a given pixel at the destination or source.

Figure 1-2. Block Diagram and Data Paths of the BLT Engine



The command packets can specify the monochrome source or the monochrome pattern data as a pixel mask. When this feature is used, the bits that carry a value of 0 cause the bytes of the corresponding pixel at the destination to not be written to by the BLT engine, thereby preserving whatever data was originally carried within those bytes. This feature can be used in writing characters to the display, while also preserving the pre-existing backgrounds behind those characters. When both operands are in the transparent mode, the logical AND of the 2 operands are used for the write enables per pixel.

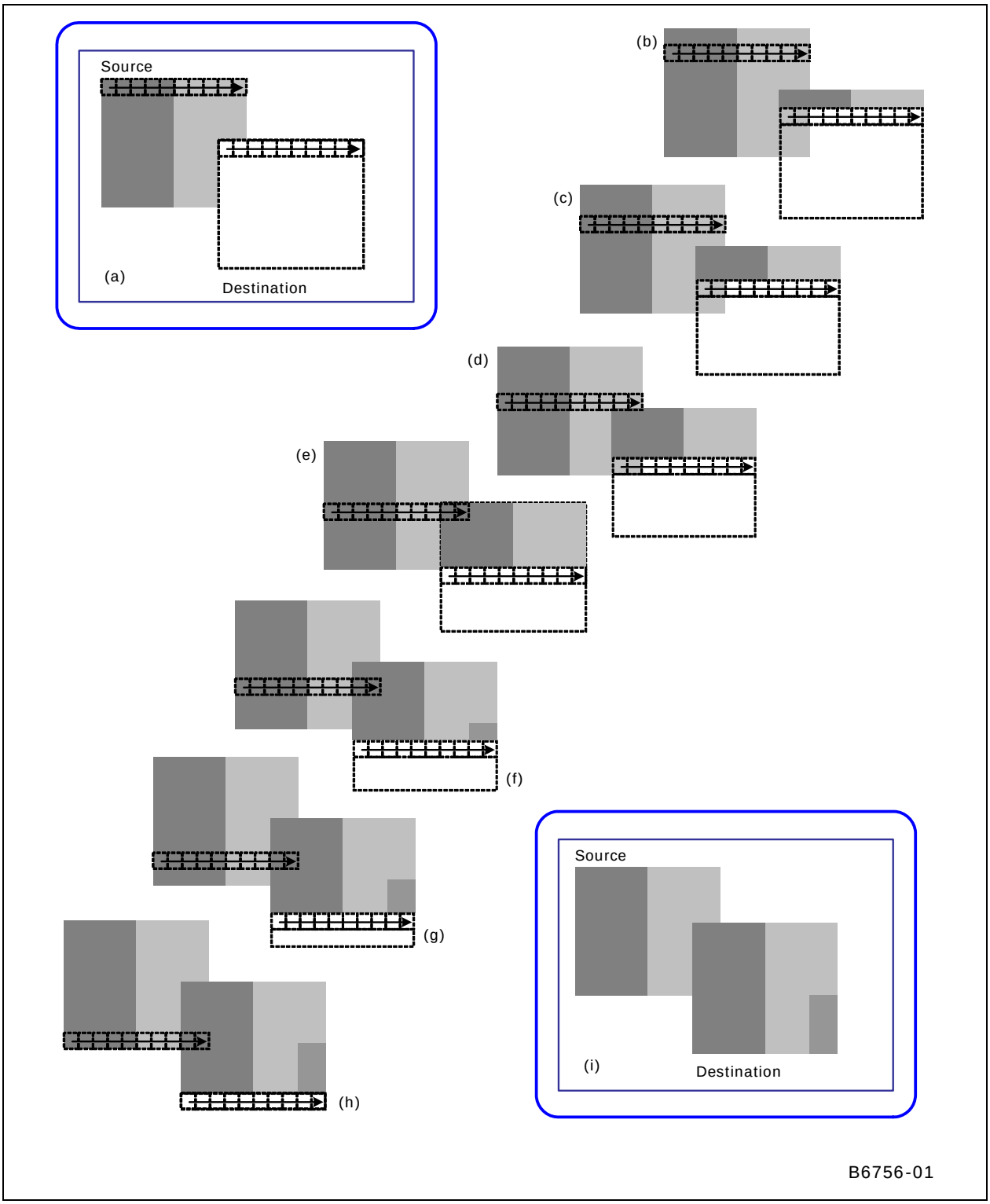
The 3-bit field, destination transparency mode, within the command packets can select per-pixel write-masking with a mask based on the results of color comparisons. The monochrome source background and foreground are range compared with either the bytes for the pixels at the destination or the source operand. This operation is described in the BLT command packet and register descriptions.

1.2.1.5 When the Source and Destination Locations Overlap

It is possible to have BLT operations in which the locations of the source and destination data overlap. This frequently occurs in BLT operations where a user is shifting the position of a graphical item on the display by only a few pixels. In these situations, the BLT engine must be programmed so that destination data is not written into destination locations that overlap with source locations before the source data at those locations has been read. Otherwise, the source data will become corrupted. The XY commands determine whether there is an overlap and perform the accesses in the proper direction to avoid data corruption.

The following figure shows how the source data can be corrupted when a rectangular block is copied from a source location to an overlapping destination location. The BLT engine typically reads from the source location and writes to the destination location starting with the left-most pixel in the top-most line of both, as shown in step (a). As shown in step (b), corruption of the source data has already started with the copying of the top-most line in step (a) — part of the source that originally contained lighter-colored pixels has now been overwritten with darker-colored pixels. More source data corruption occurs as steps (b) through (d) are performed. At step (e), another line of the source data is read, but the two right-most pixels of this line are in the region where the source and destination locations overlap, and where the source has already been overwritten as a result of the copying of the top-most line in step (a). Starting in step (f), darker-colored pixels can be seen in the destination where lighter-colored pixels should be. This errant effect occurs repeatedly throughout the remaining steps in this BLT operation. As more lines are copied from the source location to the destination location, it becomes clear that the end result is not what was originally intended.

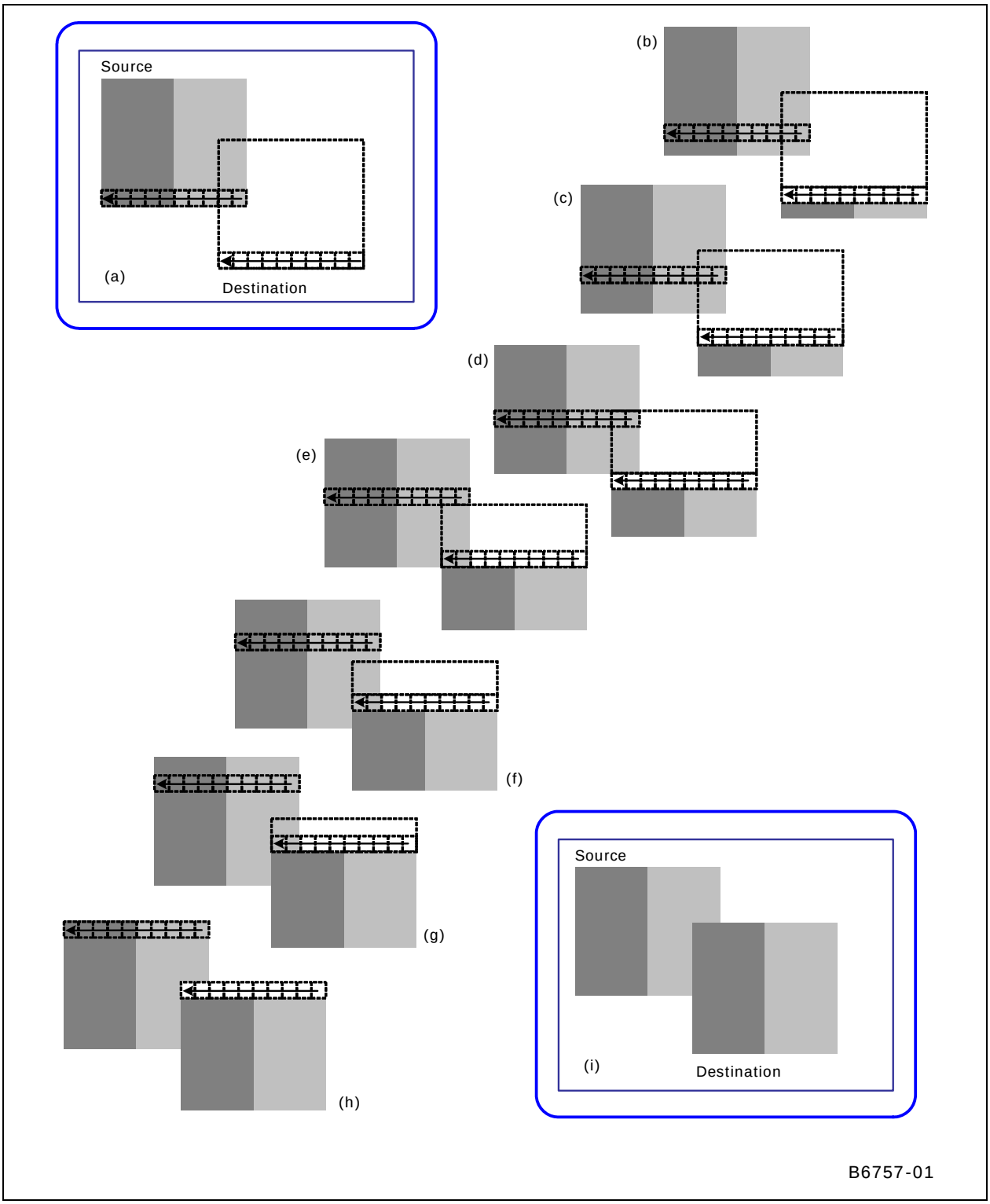
Figure 1-3. Source Corruption in BLT with Overlapping Source and Destination Locations



The BLT engine can alter the order in which source data is read and destination data is written when necessary to avoid source data corruption problems when the source and destination locations overlap. The command packets provide the ability to change the point at which the BLT engine begins reading and writing data from the upper left-hand corner (the usual starting point) to one of the other three corners. The BLT engine may be set to read data from the source and write it to the destination starting at any of the four corners of the panel.

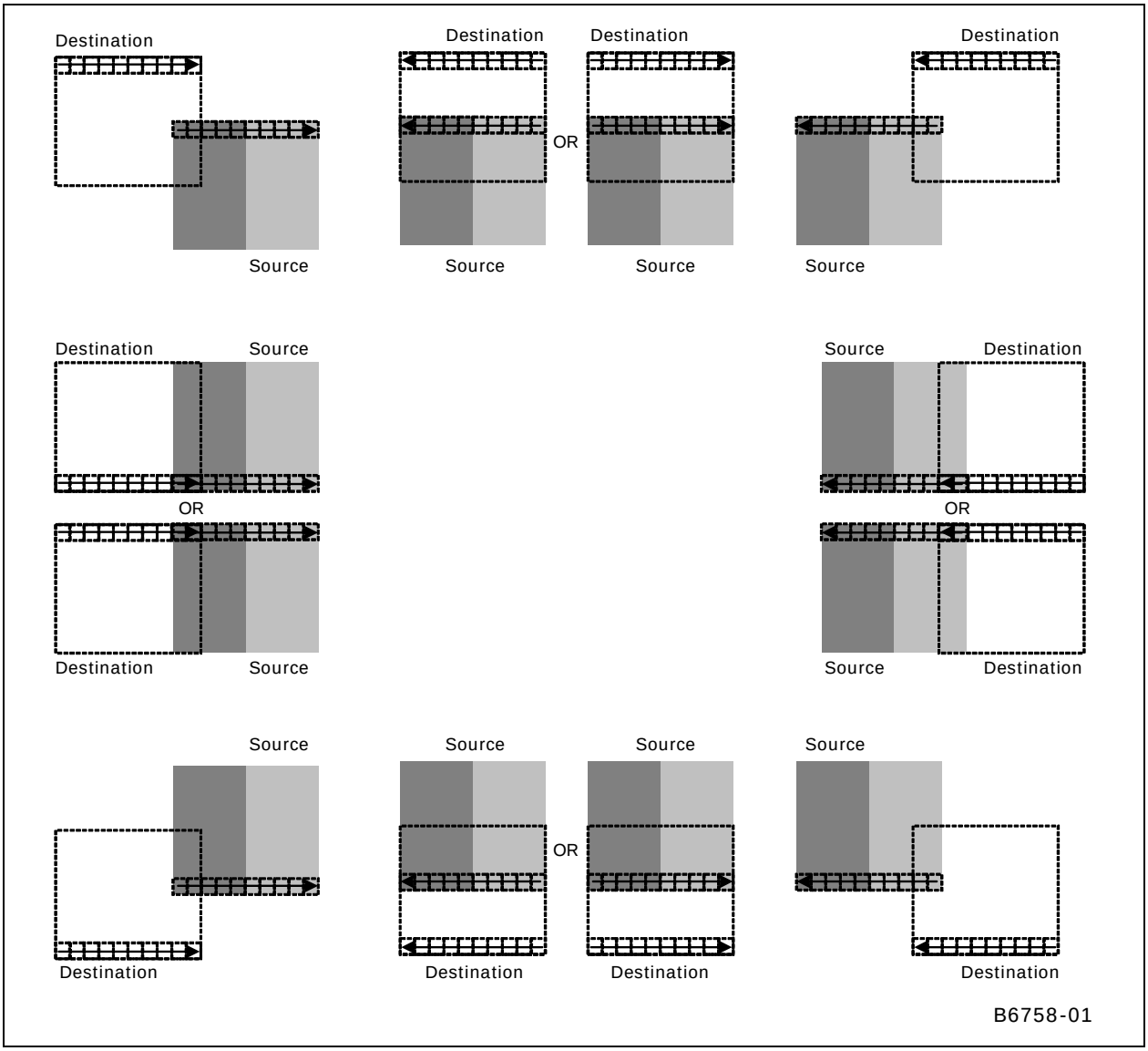
The XY command packets perform the necessary comparisons and start at the proper corner of each operand which avoids data corruption.

Figure 1-4. Correctly Performed BLT with Overlapping Source and Destination Locations



The following figure illustrates how this feature of the BLT engine can be used to perform the same BLT operation as was illustrated in the figure above, while avoiding the corruption of source data. As shown in the figure below, the BLT engine reads the source data and writes the data to the destination starting with the right-most pixel of the bottom-most line. By doing this, no pixel existing where the source and destination locations overlap will ever be written to before it is read from by the BLT engine. By the time the BLT operation has reached step (e) where two pixels existing where the source and destination locations overlap are about to be over written, the source data for those two pixels has already been read.

Figure 1-5. Suggested Starting Points for Possible Source and Destination Overlap Situations



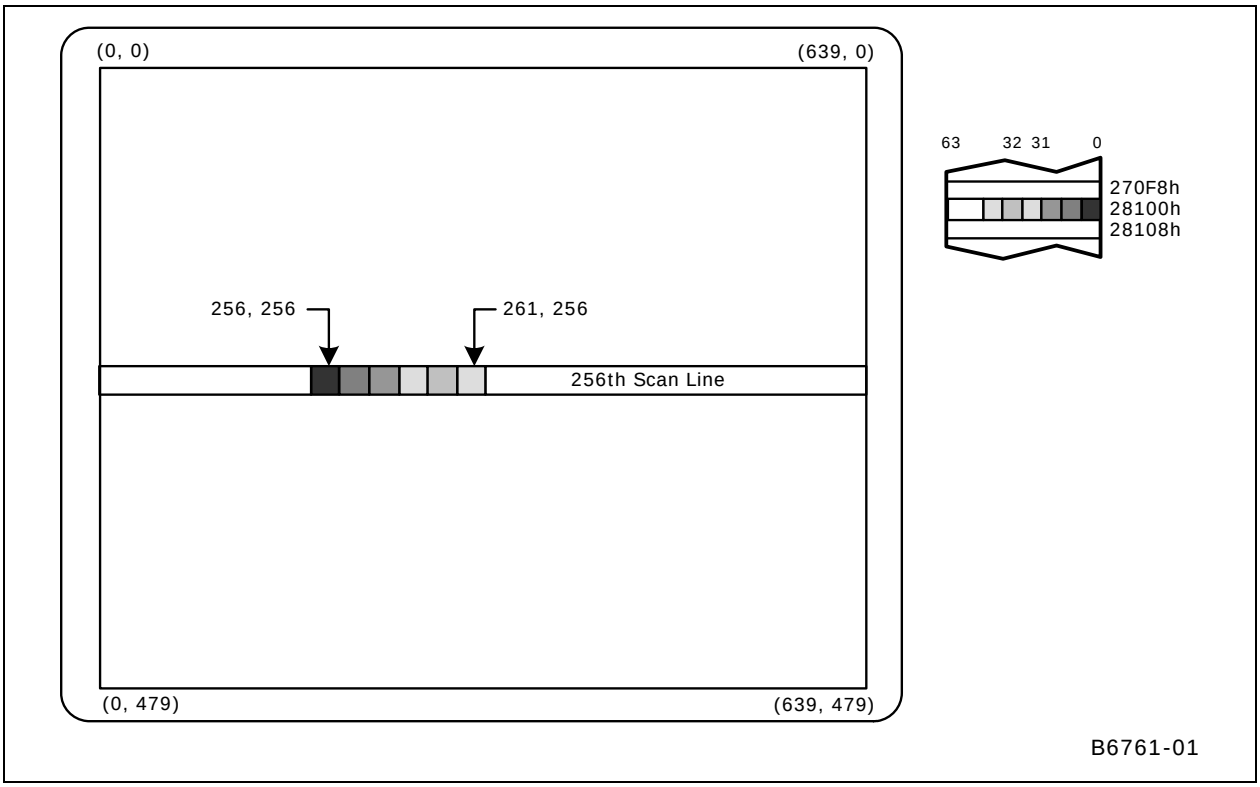
The figure above shows the recommended lines and pixels to be used as starting points in each of 8 possible ways in which the source and destination locations may overlap. In general, the starting point should be within the area in which the source and destination overlap.

1.2.2 Basic Graphics Data Considerations

1.2.2.1 Contiguous vs. Discontinuous Graphics Data

Graphics data stored in memory, particularly in the frame buffer of a graphics system, has organizational characteristics that often distinguish it from other varieties of data. The main distinctive feature is the tendency for graphics data to be organized in a discontinuous block of graphics data made up of multiple sub-blocks of bytes, instead of a single contiguous block of bytes.

Figure 1-6. Representation of On-Screen Single 6-Pixel Line in the Frame Buffer

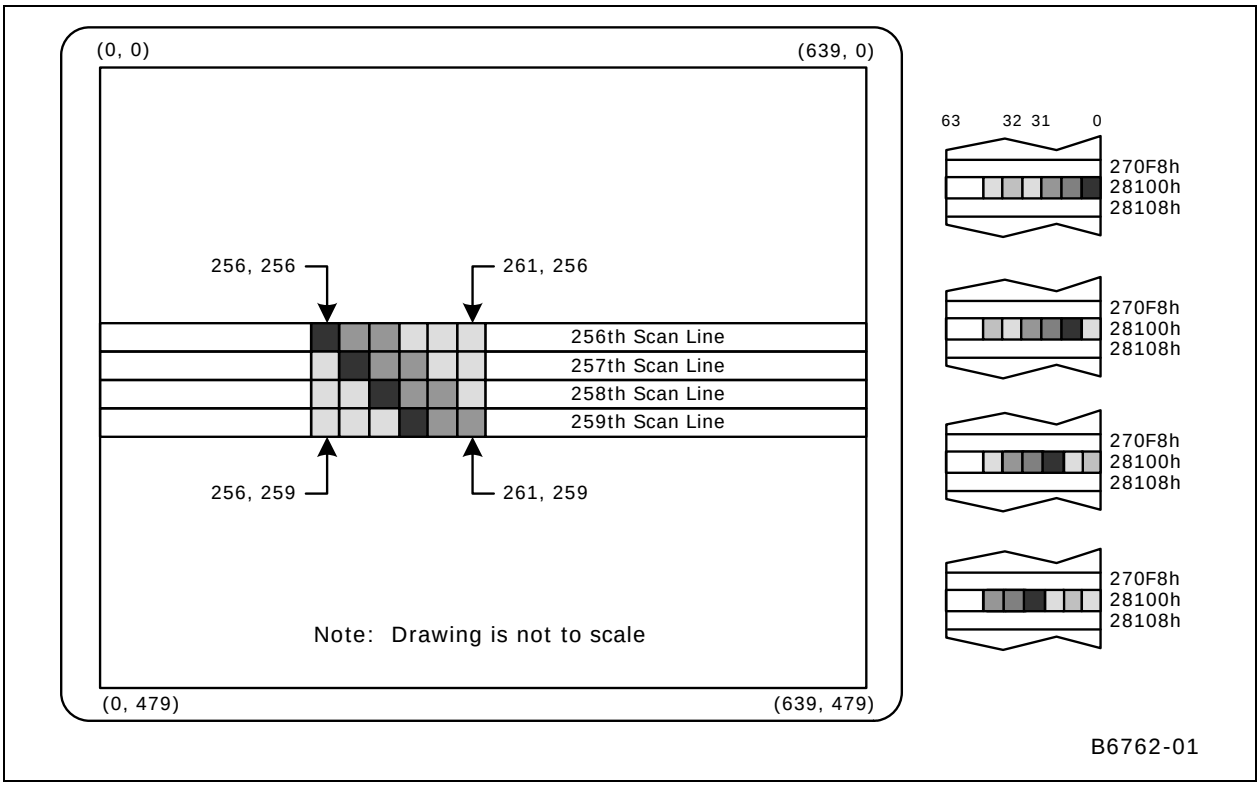


The figure above shows an example of contiguous graphics data — a horizontal line made up of six adjacent pixels within a single scan line on a display with a resolution of 640x480. Presuming that the graphics system driving this display has been set to 8 bits per pixel and that the frame buffer's starting address of 0h corresponds to the upper left-most pixel of this display, then the six pixels that make this horizontal line starting at coordinates (256, 256) occupies the six bytes starting at frame buffer address 28100h, and ending at address 28105h.

In this case, there is only one scan line's worth of graphics data in this single horizontal line, so the block of graphics data for all six of these pixels exists as a single, contiguous block comprised of only these six bytes. The starting address and the number of bytes are the only pieces of information that a BLT engine would require to read this block of data.

The simplicity of the above example of a single horizontal line contrasts sharply to the example of discontinuous graphics data depicted in the figure below. The simple six-pixel line of the figure above is now accompanied by three more six-pixel lines placed on subsequent scan lines, resulting in the 6x4 block of pixels shown.

Figure 1-7. Representation of On-Screen 6x4 Array of Pixels in the Frame Buffer



Since there are other pixels on each of the scan lines on which this 6x4 block exists that are not part of this 6x4 block, what appears to be a single 6x4 block of pixels on the display must be represented by a discontinuous block of graphics data made up of 4 separate sub-blocks of six bytes apiece in the frame buffer at addresses 28100h, 28380h, 28600h, and 28880h. This situation makes the task of reading what appears to be a simple 6x4 block of pixels more complex. However, there are two characteristics of this 6x4 block of pixels that help simplify the task of specifying the locations of all 24 bytes of this discontinuous block of graphics data: all four of the sub-blocks are of the same length, and the four sub-blocks are separated from each other at equal intervals.

The BLT engine is designed to make use of these characteristics of graphics data to simplify the programming required to handle discontinuous blocks of graphics data. For such a situation, the BLT engine requires only four pieces of information: the starting address of the first sub-block, the length of a sub-block, the offset (in bytes), pitch, of the starting address of each subsequent sub-block, and the quantity of sub-blocks.

1.2.2.2 Source Data

The source data may exist in the frame buffer or elsewhere in the graphics aperture where the BLT engine may read it directly, or it may be provided to the BLT engine by the host CPU through the command packets. The block of source graphics data may be either contiguous or discontinuous, and may be either in color (with a color depth that matches that to which the BLT engine has been set) or monochrome.

The source select bit in the command packets specifies whether the source data exists in the frame buffer or is provided through the command packets. Monochrome source data is always specified as being supplied through an immediate command packet.

If the color source data resides within the frame buffer or elsewhere in the graphics aperture, then the Source Address Register, specified in the command packets is used to specify the address of the source.

In cases where the host CPU provides the source data, it does so by writing the source data to ring buffer directly after the BLT command that requires the data or uses an IMMEDIATE_INDIRECT_BLT command packet which has a size and pointer to the operand in Graphics aperture.

The block of bytes sent by the host CPU through the command packets must be quadword-aligned and the source data contained within the block of bytes must also be aligned.

To accommodate discontinuous source data, the source and destination pitch registers can be used to specify the offset in bytes from the beginning of one scan line's worth source data to the next. Otherwise, if the source data is contiguous, then an offset equal to the length of a scan line's worth of source data should be specified.

1.2.2.3 Monochrome Source Data

The opcode of the command packet specifies whether the source data is color or monochrome. Since monochrome graphics data only uses one bit per pixel, each byte of monochrome source data typically carries data for 8 pixels which hinders the use of byte-oriented parameters when specifying the location and size of valid source data. Some additional parameters must be specified to ensure the proper reading and use of monochrome source data by the BLT engine. The BLT engine also provides additional options for the manipulation of monochrome source data versus color source data.

The various bit-wise logical operations and per-pixel write-masking operations were designed to work with color data. In order to use monochrome data, the BLT engine converts it into color through a process called color expansion, which takes place as a BLT operation is performed. In color expansion the single bits of monochrome source data are converted into one, two, or four bytes (depending on the color depth) of color data that are set to carry value corresponding to either the foreground or background color that have been specified for use in this conversion process. If a given bit of monochrome source data carries a value of 1, then the byte(s) of color data resulting from the conversion process will be set to carry the value of the foreground color. If a given bit of monochrome source data carries a value of 0, then the resulting byte(s) will be set to the value of the background color. The foreground and background colors used in the color expansion of monochrome source data can be set in the source expansion foreground color register and the source expansion background color register.

The BLT Engine requires that the bit alignment of each scan line's worth of monochrome source data be specified. Each scan line's worth of monochrome source data is word aligned but can actually start on any bit boundary of the first byte. Monochrome text is special cased and it is bit or byte packed, where in bit packed there are no invalid pixels (bits) between scan lines. There is a 3 bit field which indicates the starting pixel position within the first byte for each scan line, Mono Source Start.

The BLT engine also provides various clipping options for use with specific BLT commands (BLT_TEXT) with a monochrome source. Clipping is supported through: Clip rectangle Y addresses or coordinates and X coordinates along with scan line starting and ending addresses (with Y addresses) along with X starting and ending coordinates.

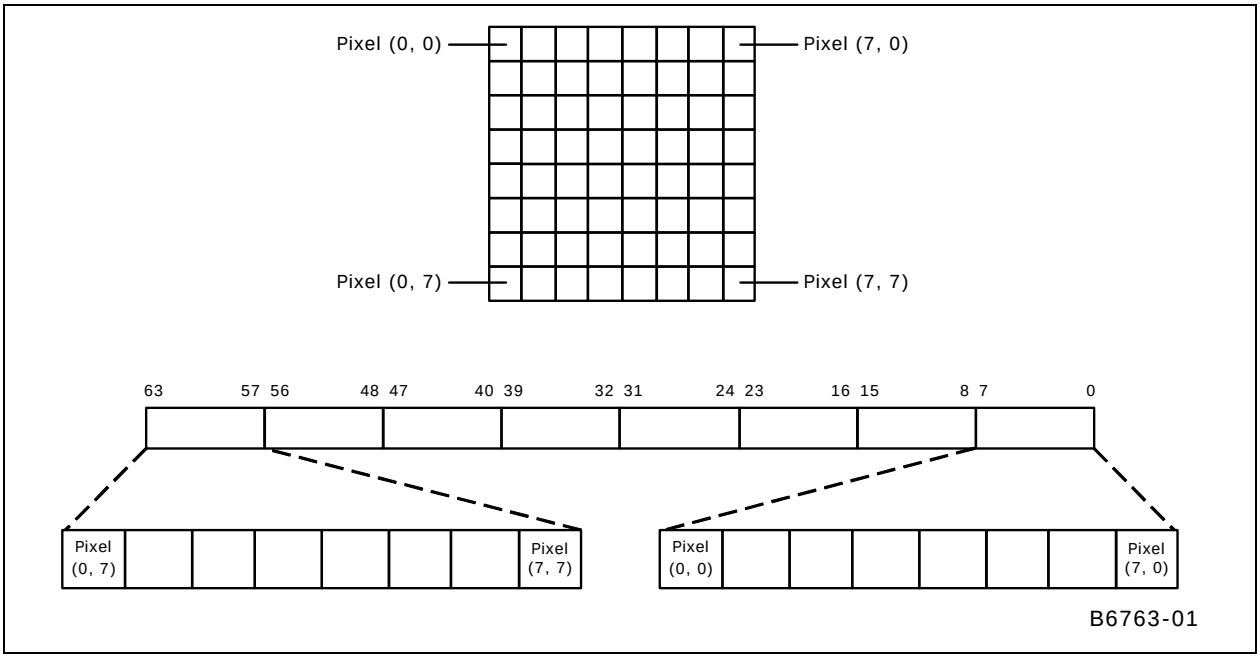
The maximum immediate source size is 128 bytes.

1.2.2.4 Pattern Data

The color pattern data must exist within the frame buffer or Graphics aperture where the BLT engine may read it directly or it can be sent through the command stream. The pattern data must be located in linear memory. Monochrome pattern data is supplied by the command packet when it is to be used. As shown in figure below, the block of pattern graphics data always represents a block of 8x8 pixels. The bits or bytes of a block of pattern data may be organized in the frame buffer memory in only one of three ways, depending upon its color depth which may be 8, 16, or 32 bits per pixel (whichever matches the color depth to which the BLT engine has been set), or monochrome.

The maximum color pattern size is 256 bytes.

Figure 1-8. Pattern Data -- Always an 8x8 Array of Pixels



The Pattern Address Register is used to specify the address of the color pattern data at which the block of pattern data begins. The three least significant bits of the address written to this register are ignored, because the address must be in terms of quadwords. This is because the pattern must always be located on an address boundary equal to its size. Monochrome patterns take up 8 bytes, or a single quadword of space, and are loaded through the command packet that uses it. Similarly, color patterns with color depths of 8, 16, and 32 bits per pixel must start on 64-byte, 128-byte and 256-byte boundaries, respectively. The next 3 figures show how monochrome, 8bpp, 16bpp, and 32bpp pattern data , respectively, is organized in memory.

Figure 1-9. 8bpp Pattern Data -- Occupies 64 Bytes (8 quadwords)

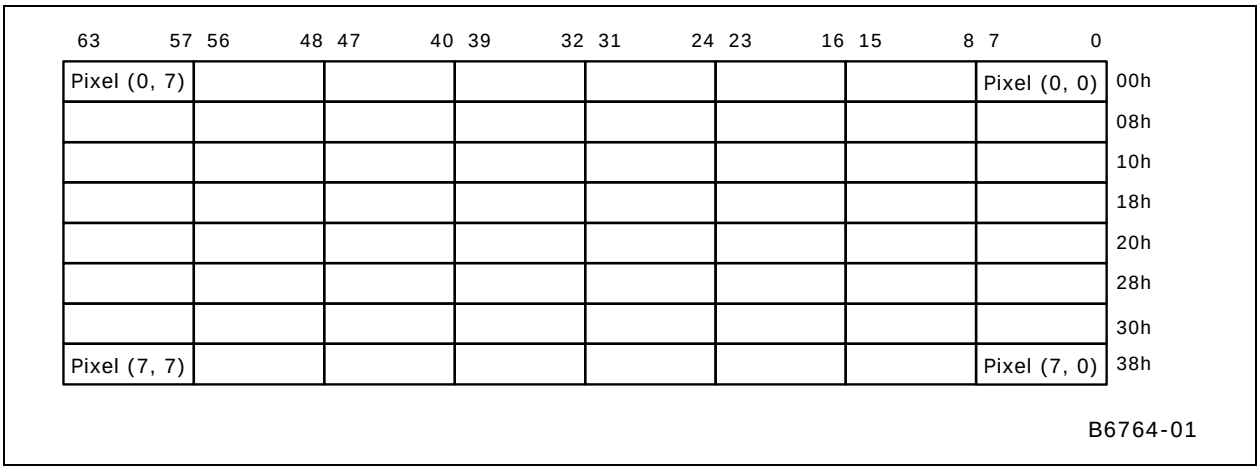


Figure 1-10. 16bpp Pattern Data -- Occupies 128 Bytes (16 quadwords)

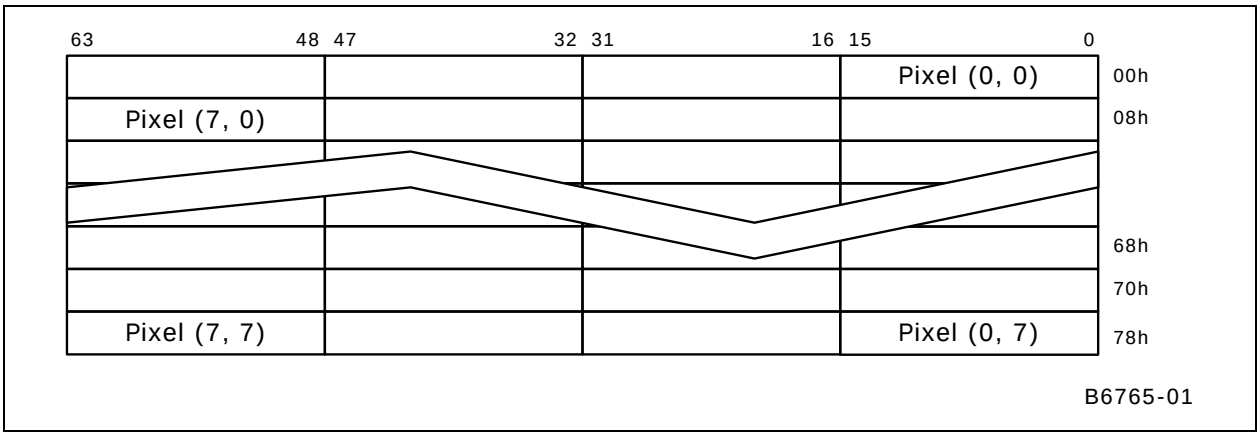
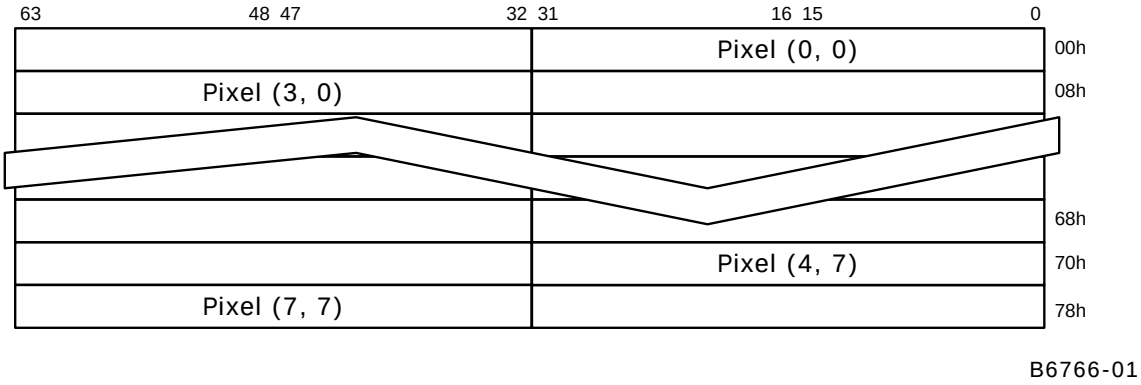


Figure 1-11. 32bpp Pattern Data -- Occupies 256 Bytes (32 quadwords)



The opcode of the command packet specifies whether the pattern data is color or monochrome. The various bit-wise logical operations and per-pixel write-masking operations were designed to work with color data. In order to use monochrome pattern data, the BLT engine is designed to convert it into color

through a process called “color expansion” which takes place as a BLT operation is performed. In color expansion, the single bits of monochrome pattern data are converted into one, two, or four bytes (depending on the color depth) of color data that are set to carry values corresponding to either the foreground or background color that have been specified for use in this process. The foreground color is used for pixels corresponding to a bit of monochrome pattern data that carry the value of 1, while the background color is used where the corresponding bit of monochrome pattern data carries the value of 0. The foreground and background colors used in the color expansion of monochrome pattern data can be set in the Pattern Expansion Foreground Color Register and Pattern Expansion Background Color Register.

1.2.2.5 Destination Data

There are actually two different types of “destination data”: the graphics data already residing at the location that is designated as the destination, and the data that is to be written into that very same location as a result of a BLT operation.

The location designated as the destination must be within the frame buffer or Graphics aperture where the BLT engine can read from it and write to it directly. The blocks of destination data to be read from and written to the destination may be either contiguous or discontinuous. All data written to the destination will have the color depth to which the BLT engine has been set. It is presumed that any data already existing at the destination which will be read by the BLT engine will also be of this same color depth — the BLT engine neither reads nor writes monochrome destination data.

The Destination Address Register is used to specify the address of the destination.

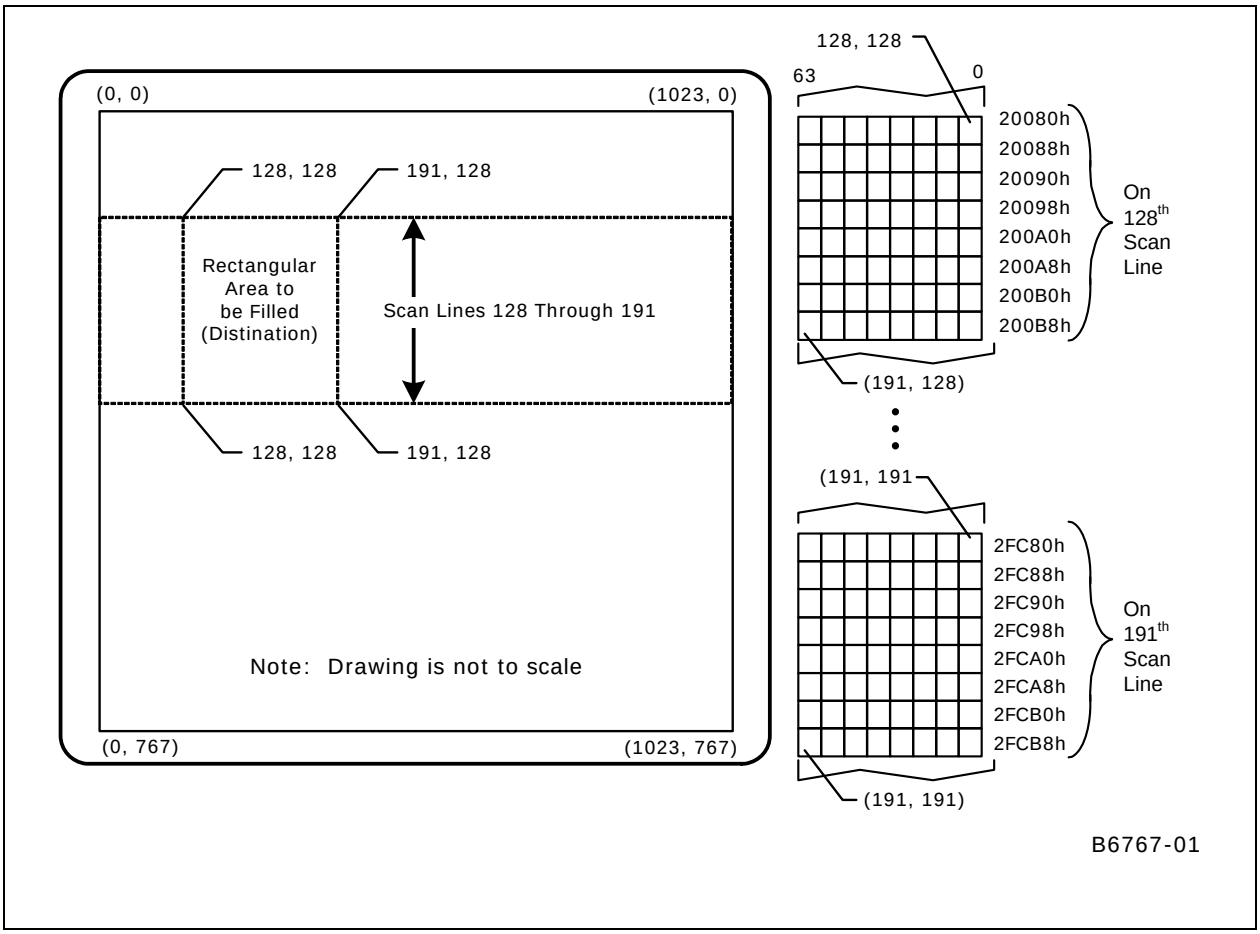
To accommodate discontinuous destination data, the Source and Destination Pitch Registers can be used to specify the offset in bytes from the beginning of one scan line’s worth of destination data to the next. Otherwise, if the destination data is contiguous, then an offset equal to the length of a scan line’s worth of destination data should be specified.

1.2.3 BLT Programming Examples

1.2.3.1 Pattern Fill — A Very Simple BLT

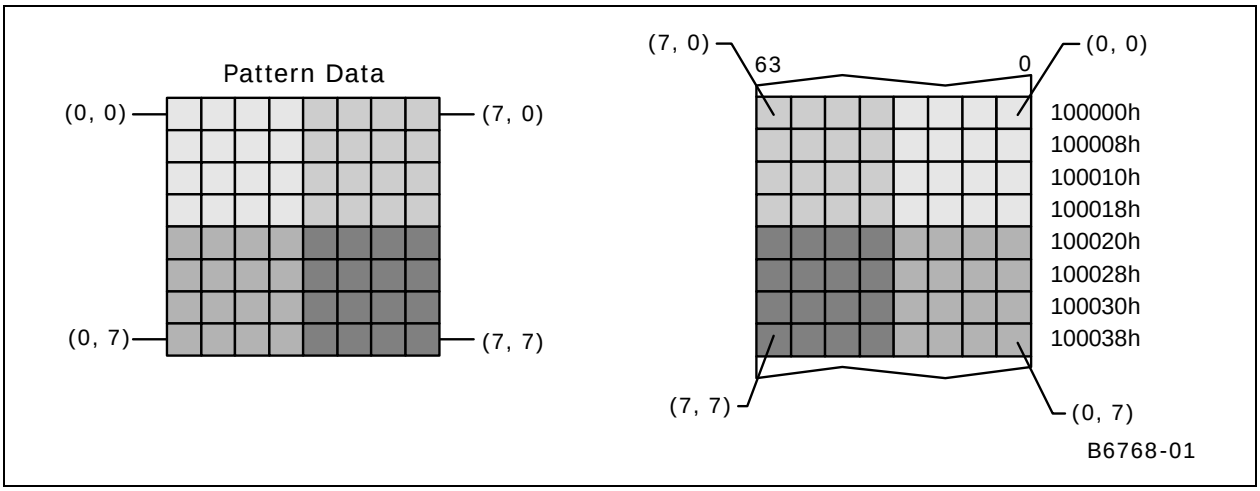
In this example, a rectangular area on the screen is to be filled with a color pattern stored as pattern data in off-screen memory. The screen has a resolution of 1024x768 and the graphics system has been set to a color depth of 8 bits per pixel.

Figure 1-12. On-Screen Destination for Example Pattern Fill BLT



As shown in the figure above, the rectangular area to be filled has its upper left-hand corner at coordinates (128, 128) and its lower right-hand corner at coordinates (191, 191). These coordinates define a rectangle covering 64 scan lines, each scan line's worth of which is 64 pixels in length — in other words, an array of 64x64 pixels. Presuming that the pixel at coordinates (0, 0) corresponds to the byte at address 00h in the frame buffer memory, the pixel at (128, 128) corresponds to the byte at address 20080h.

Figure 1-13. Pattern Data for Example Pattern Fill BLT



As shown in figure above, the pattern data occupies 64 bytes starting at address 100000h. As always, the pattern data represents an 8x8 array of pixels.

The BLT command packet is used to select the features to be used in this BLT operation, and must be programmed carefully. The vertical alignment field should be set to 0 to select the top-most horizontal row of the pattern as the starting row used in drawing the pattern starting with the top-most scan line covered by the destination. The pattern data is in color with a color depth of 8 bits per pixel, so the dynamic color enable should be asserted with the dynamic color depth field should be set to 0. Since this BLT operation does not use per-pixel write-masking (destination transparency mode), this field should be set to 0. Finally, the raster operation field should be programmed with the 8-bit value of F0h to select the bit-wise logical operation in which a simple copy of the pattern data to the destination takes place. Selecting this bit-wise operation in which no source data is used as an input causes the BLT engine to automatically forego either reading source data from the frame buffer.

The Destination Pitch Register must be programmed with number of bytes in the interval from the start of one scan line's worth of destination data to the next. Since the color depth is 8 bits per pixel and the horizontal resolution of the display is 1024, the value to be programmed into these bits is 400h, which is equal to the decimal value of 1024.

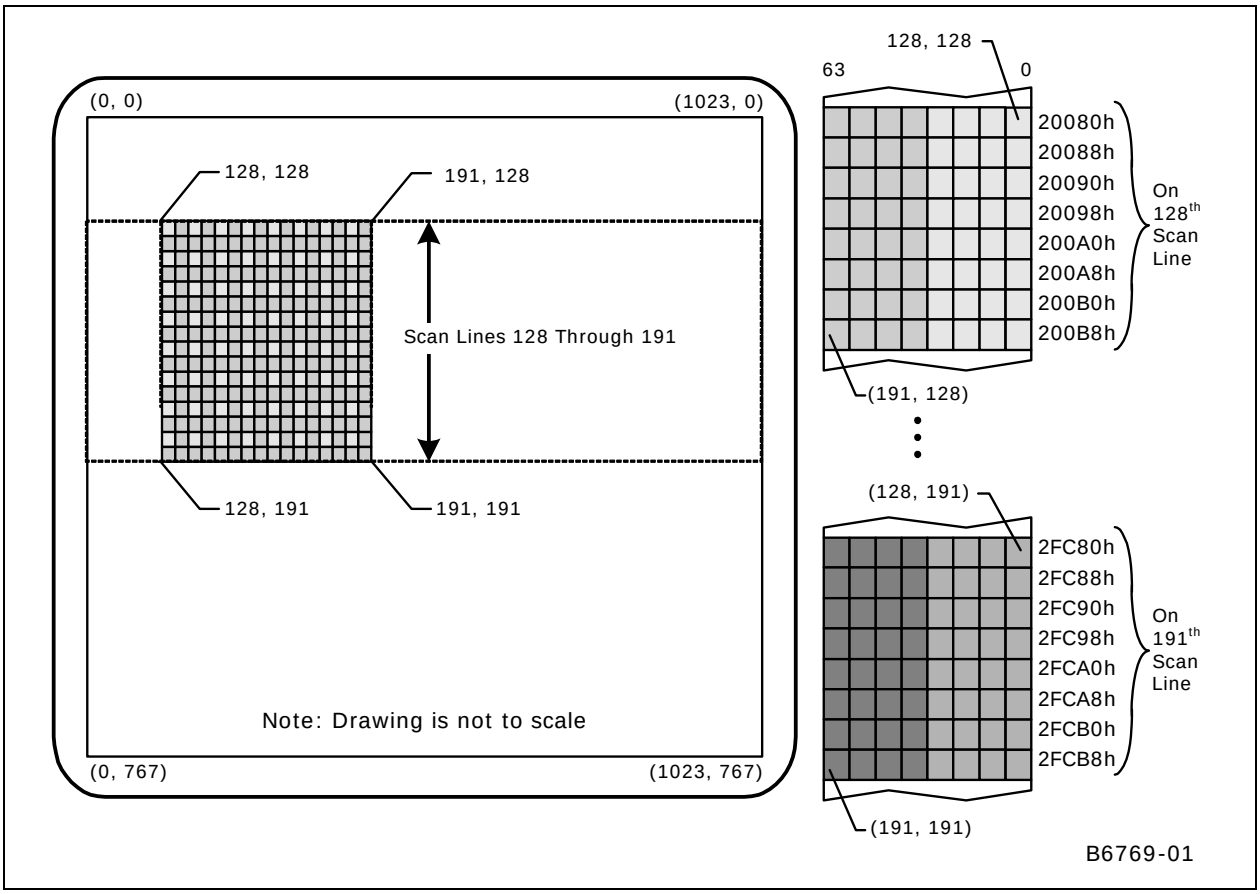
Bits [31:3] of the Pattern Address Register must be programmed with the address of the pattern data.

Similarly, bits [31:0] of the Destination Address Register must be programmed with the byte address at the destination that will be written to first. In this case, the address is 20080h, which corresponds to the byte representing the pixel at coordinates (128, 128).

This BLT operation does not use the values in the Source Address Register or the Source Expansion Background or Foreground Color Registers.

The Destination Width and Height Registers (or the Destination X and Y Coordinates) must be programmed with values that describe to the BLT engine the 64x64 pixel size of the destination location. The height should be set to carry the value of 40h, indicating that the destination location covers 64 scan lines. The width should be set to carry the value of 40h, indicating that each scan line's worth of destination data occupies 64 bytes. All of this information is written to the ring buffer using the PAT_BLT (or XY_PAT_BLT) command packet.

Figure 1-14. Results of Example Pattern Fill BLT

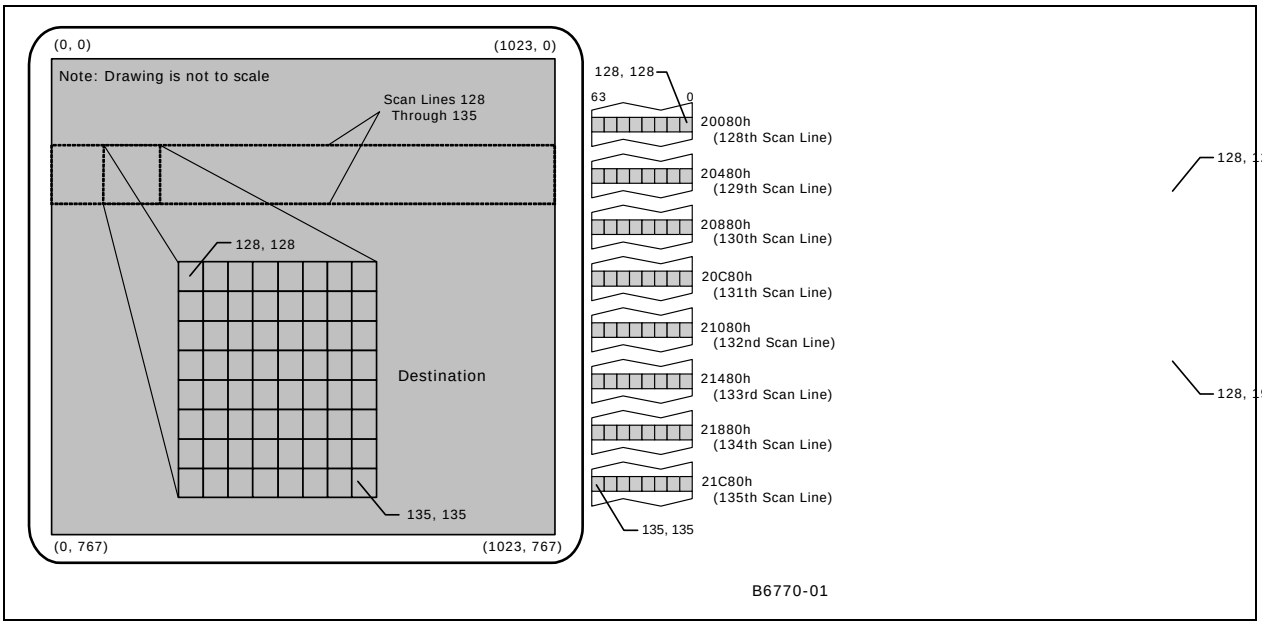


The figure above shows the end result of performing this BLT operation. The 8x8 pattern has been repeatedly copied ("tiled") into the entire 64x64 area at the destination.

1.2.3.2 Drawing Characters Using a Font Stored in System Memory

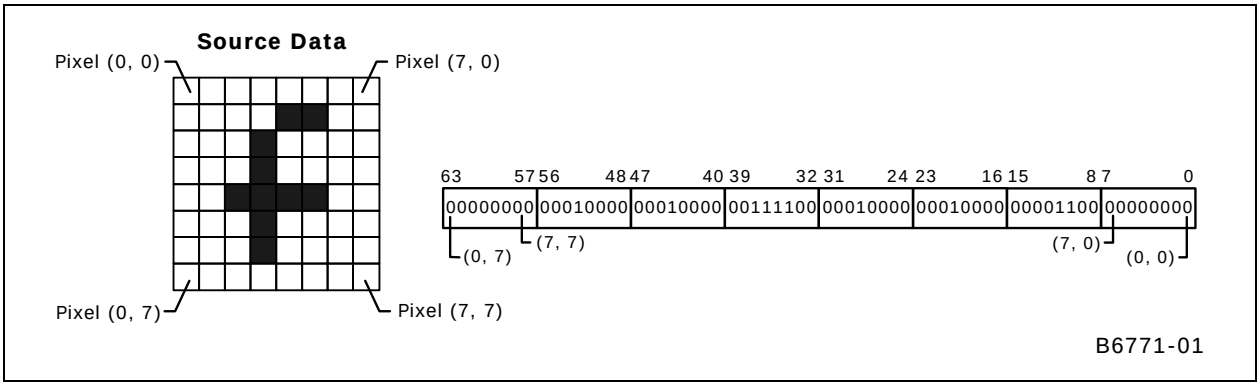
In this example BLT operation, a lowercase letter "f" is to be drawn in black on a display with a gray background. The resolution of the display is 1024x768, and the graphics system has been set to a color depth of 8 bits per pixel.

Figure 1-15. On-Screen Destination for Example Character Drawing BLT



The figure above shows the display on which this letter “f” is to be drawn. As shown in this figure, the entire display has been filled with a gray color. The letter “f” is to be drawn into an 8x8 region on the display with the upper left-hand corner at the coordinates (128, 128).

Figure 1-16. Source Data in System Memory for Example Character Drawing BLT



The figure above shows both the 8x8 pattern making up the letter “f” and how it is represented somewhere in the host’s system memory — the actual address in system memory is not important. The letter “f” is represented in system memory by a block of monochrome graphics data that occupies 8 bytes. Each byte carries the 8 bits needed to represent the 8 pixels in each scan line’s worth of this graphics data. This type of pattern is often used to store character fonts in system memory.

During this BLT operation, the host CPU will read this representation of the letter “f” from system memory, and write it to the BLT engine by performing memory writes to the ring buffer as an immediate monochrome BLT operand following the BLT_TEXT command. The BLT engine will receive this data through the command stream and use it as the source data for this BLT operation. The BLT engine will be set to the same color depth as the graphics system — 8 bits per pixel, in this case. Since the source data

in this BLT operation is monochrome, color expansion must be used to convert it to an 8 bpp color depth. To ensure that the gray background behind this letter “f” is preserved, per-pixel write masking will be performed, using the monochrome source data as the pixel mask.

The BLT Setup and Text_immediate command packets are used to select the features to be used in this BLT operation. Only the fields required by these two command packets must be programmed carefully. The BLT engine ignores all other registers and fields. The source select field in the Text_immediate command must be set to 1, to indicate that the source data is provided by the host CPU through the command packet. Finally, the raster operation field should be programmed with the 8-bit value CCh to select the bit-wise logical operation that simply copies the source data to the destination. Selecting this bit-wise operation in which no pattern data is used as an input, causes the BLT engine to automatically forego reading pattern data from the frame buffer.

The Setup Pattern/Source Expansion Foreground Color Register to specify the color with which the letter “f” will be drawn. There is no Source address. All scan lines of the glyph are bit packed and the clipping is controlled by the ClipRect registers from the SETUP_BLT command and the Destination Y1, Y2, X1, and X2 registers in the TEXT_BLT command. Only the pixels that are within (inclusive comparisons) the clip rectangle are written to the destination surface.

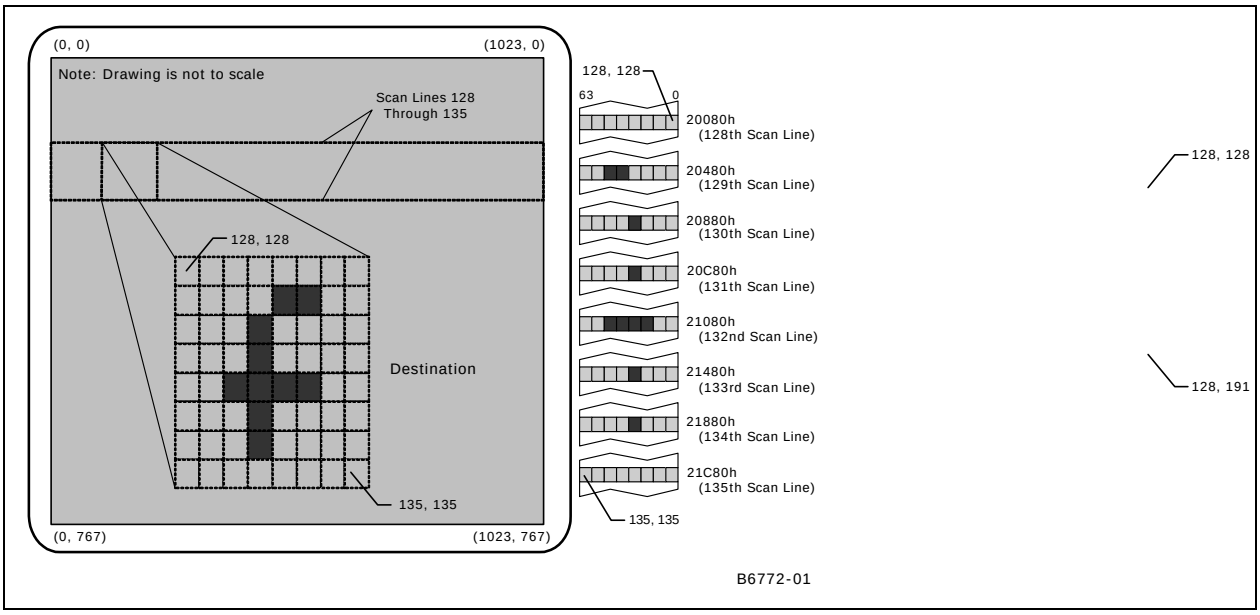
The Destination Pitch Register must be programmed with a value equal to the number of bytes in the interval between the first bytes of each adjacent scan line’s worth of destination data. Since the color depth is 8 bits per pixel and the horizontal resolution of the display is 1024 pixels, the value to be programmed into these bits is 400h, which is equal to the decimal value of 1024. Since the source data used in this BLT operation is monochrome, the BLT engine will not use a byte-oriented pitch value for the source data.

Since the source data is monochrome, color expansion is required to convert it to color with a color depth of 8 bits per pixel. Since the Setup Pattern/Source Expansion Foreground Color Register is selected to specify the foreground color of black to be used in drawing the letter “f”, this register must be programmed with the value for that color. With the graphics system set for a color depth of 8 bits per pixel, the actual colors are specified in the RAMDAC palette, and the 8 bits stored in the frame buffer for each pixel actually specify the index used to select a color from that palette. This example assumes that the color specified at index 00h in the palette is black, and therefore bits [7:0] of this register should be set to 00h to select black as the foreground color. The BLT engine ignores bits [31:8] of this register because the selected color depth is 8 bits per pixel. Even though the color expansion being performed on the source data normally requires that both the foreground and background colors be specified, the value used to specify the background color is not important in this example. Per-pixel write-masking is being performed with the monochrome source data as the pixel mask, which means that none of the pixels in the source data that will be converted to the background color will ever be written to the destination. Since these pixels will never be seen, the value programmed into the Pattern/Source Expansion Background Color Register to specify a background color is not important.

The Destination Width and Height Registers are not used. The Y1, Y2, X1, and X2 are used to describe to the BLT engine the 8x8 pixel size of the destination location. The Destination Y1 and Y2 address (or coordinate) registers must be programmed with the starting and ending scan line address (or Y coordinates) of the destination data. This address is specified as an offset from the start of the frame buffer of the scan line at the destination that will be written to first. The destination X1 and X2 registers must be programmed with the starting and ending pixel offsets from the beginning of the scan line.

This BLT operation does not use the values in the Pattern Address Register, the Source Expansion Background Color Register, or the Source Expansion Foreground Color Register.

Figure 1-17. Results of Example Character Drawing BLT



The preceding shows the end result of performing this BLT operation. Only the pixels that form part of the actual letter “f” have been drawn into the 8x8 destination location on the display, leaving the other pixels within the destination with their original gray color.

1.3 BLT Instruction Overview

This chapter defines the instructions used to control the 2D (BLT) rendering function.

The instructions detailed in this chapter are used across devices. However, slight changes may be present in some instructions (i.e., for features added or removed), or some instructions may be removed entirely. Refer to the *Device Dependencies* chapter for summary information regarding device-specific behaviors/interfaces/features.

The XY instructions offload the drivers by providing X and Y coordinates and taking care of the access directions for overlapping BLTs without fields specified by the driver.

Color pixel sizes supported are 8, 16, and 32 bits per pixel (bpp). All pixels are naturally aligned.

1.4 BLT Engine State

Most of the BLT instructions are state-free, which means that all states required to execute the command is within the instruction. If clipping is not used, then there is no shared state for many of the BLT instructions. This allows the BLT Engine to be shared by many drivers with minimal synchronization between the drivers.

Instructions which share state are:

All instructions that are X,Y commands and use the Clipping Rectangle by asserting the Clip Enable field

All XY_Setup Commands (XY_SETUP_BLT and XY_SETUP_MONO_PATTERN_SL_BLT, XY_SETUP_CLIP_BLT) load the shared state for the following commands:

XY_PIXEL_BLT	(Negative Stride (=Pitch) Not Allowed)
XY_SCANLINES_BLT	
XY_TEXT_BLT	(Negative Stride (=Pitch) Not Allowed)
XY_TEXT_IMMEDIATE_BLT	(Negative Stride (=Pitch) Not Allowed)

State registers that are saved and restored in the Logical Context:

BR1+	Setup Control (Solid Pattern Select, Clipping Enable, Mono Source Transparency Mode, Mono Pattern Transparency Mode, Color Depth[1:0], Raster Operation[7:0], & Destination Pitch[15:0]) + 32bpp Channel Mask[1:0], Mono / Color Pattern
BR05	Setup Background Color
BR06	Setup Foreground Color
BR07	Setup Pattern Base Address
BR09	Setup Destination Base Address
BR20	DW0 for a Monochrome Pattern
BR21	DW1 for a Monochrome Pattern
BR24	ClipRectY1'X1
BR25	ClipRectY2'X2

1.5 Cacheable Memory Support

The BLT Engine can be used to transfer data between cacheable ("system") memory and uncached ("main", or "UC") graphics memory using the BLT instructions. The GTT must be properly programmed to map memory pages as cacheable or UC. Only linear-mapped (not tiled) surfaces can be mapped as cacheable.

Transfers between cacheable sources and cacheable destinations are not supported. Patterns and monochrome sources can not be located in cacheable memory.

Cacheable write operands do not snoop the processor's cache nor update memory until evicted from the render cache. Cacheable read or write operands are not snooped (nor invalidated) from either internal cache by external (processor, hublink,…) accesses.

1.6 Device Cache Coherency: Render & Texture Caches

Software must initiate cache flushes to enforce coherency between the render and texture caches, i.e., both the render and texture caches must be flushed before a BLT destination surface can be reused as a texture source. Color sources and destinations use the render cache, while patterns and monochrome sources use the texture cache.

1.7 BLT Engine Instructions

The Instruction Target field is used as an opcode by the BLT Engine state machine to qualify the control bits that are relevant for executing the instruction. The descriptions for each DWord and bit field are contained in the *BLT Engine Instruction Field Definition* section. Each DWord field is described as a register, but none of these registers can be written of read through a memory mapped location – they are internal state only.

1.7.1 Blt Programming Restrictions

Overlapping Source/Destination BLTs: The following condition must be avoided when programming the Blt engine: Linear surfaces with a cache line in scan line Y for the source stream overlapping with a cache line in scan line Y-1 for the dest stream (=> non-aligned surface pitches). The cache coherency rules combined with the Blitter data consumption rules result in UNDEFINED operation. (Note that this restriction will likely follow forward to future products due to architectural complexities.) There are two suggested software workarounds:

In order to perform coherent overlapping Blts, (a) the Source and Destination Base Address registers must hold the same value (without alignment restriction), and (b) the Source and Destination Pitch registers (BR11,BR13) must both be a multiple of 64 bytes.

If (a) isn't possible, do overlapping source copy BLTs as two blits, using a separate intermediate surface.

All reserved fields must be programmed to 0s.

When using monosource or text data (bit/byte/word aligned): do not program pixel widths greater than 32,745 pixels.

Ring Buffer Programming [Dev ILK only:WA]: Driver should make sure the first 3D command after the engine switch from BLT is not to be

1. DrawCall
2. Curbe
3. Media Object

The other way to do this is driver should always program a dummy 3D

NON-PIPELINE state following the BLT commands.

Immediate Commands [DevSNB] There must be at least 1 command after any immediate blitter commands before head == tail. This can be a simple MI_NOOP.

1.8 Fill/Move Instructions

These instructions use linear addresses with width and height. BLT clipping is not supported.

1.8.1 COLOR_BLT (Fill)

COLOR_BLT is the simplest BLT operation. It performs a color fill to the destination (with a possible ROP). The only operand is the destination operand which is written dependent on the raster operation. The solid pattern color is stored in the pattern background register.

This instruction is optimized to run at the maximum memory write bandwidth.

The typical Raster operation code = F0 which performs a copy of the pattern background register to the destination.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	Instruction Target (Opcode) : 40h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:05	**Reserved.** Note no tiling specification allowed for this non-XY blit command. Only linear blits are allowed.
	04:00	DWord Length: 03h
1 = BR13	31:26	Reserved.
	25:24	Color Depth: 00 = 8 bit color. 01 = 16 bit color (656). 10 = 16 bit color (1555). 11 = 32 bit color
	23:16	Raster Operation:
	15:00	**Destination Pitch (signed):** Destination pitch in bytes (Same as before).
2 = BR14	31:16	Destination Height (in scan lines):
	15:00	Destination Width (in bytes):
3 = BR09	31:00	**Destination Address:** Address of the first byte to be written
4 = BR16	31:00	**Solid Pattern Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]

1.8.2 SRC_COPY_BLT (Move)

This BLT instruction performs a color source copy where the only operands involved is a color source and destination of the same bit width.

The source and destination operands may overlap. The command must indicate the horizontal and vertical directions: either forward or backwards to avoid data corruption. The X direction (horizontal) field applies to both the destination and source operands. The source and destination pitches (stride) are signed.

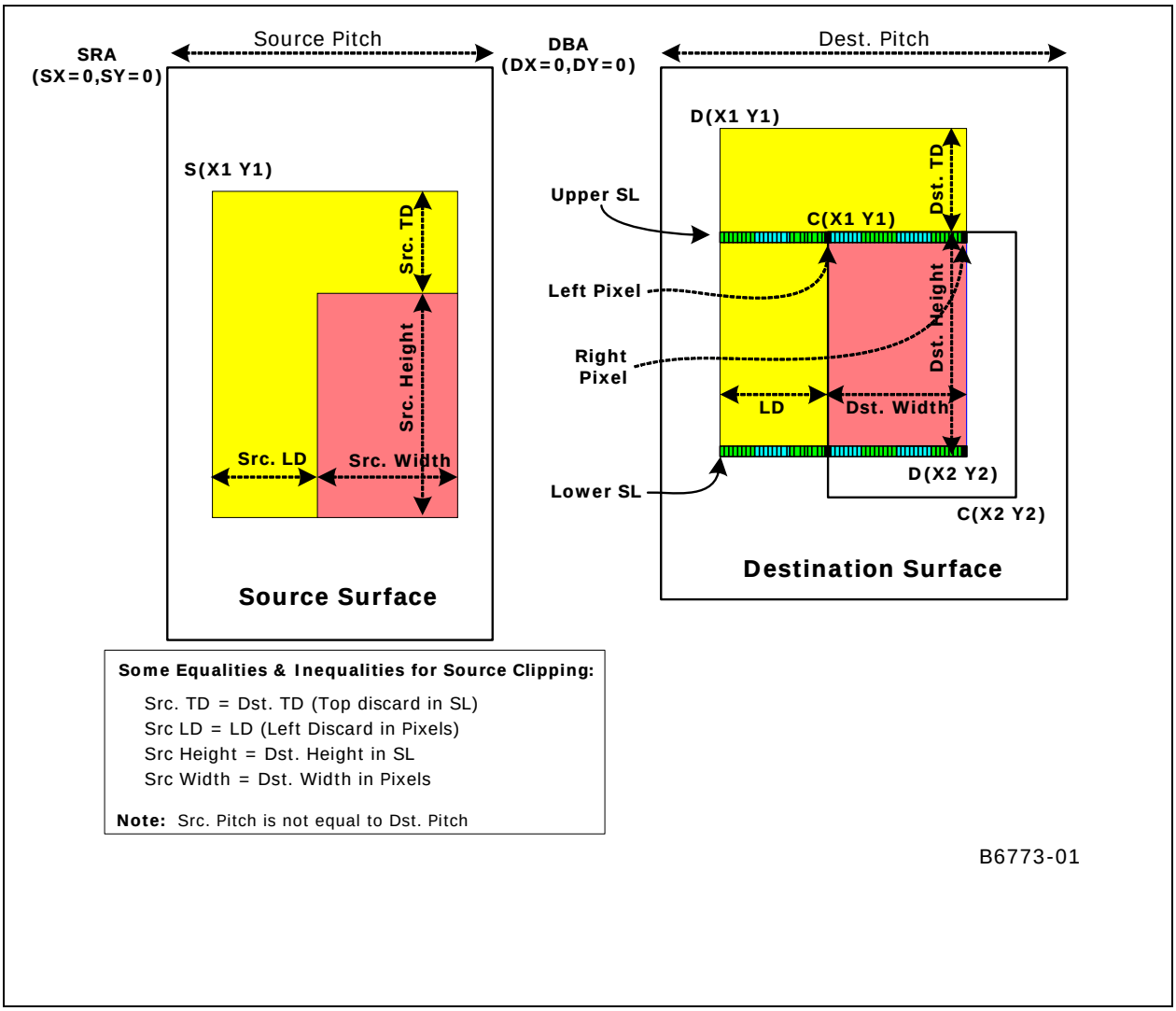
DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h – 2D Processor
	28:22	Instruction Target (Opcode) : 43h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:05	**Reserved.** Note no tiling specification allowed for this non-XY blit command. Only linear blits are allowed.
	04:00	Dword Length: 04h
1 = BR13	31	Reserved.
	30	**X Direction** (1 = written from right to left (decrementing = backwards); 0 = incrementing)
	29:26	Reserved.
	25:24	Color Depth: 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	Raster Operation:
	15:00	**Destination Pitch (signed):** Destination pitch in bytes (Same as before).
2 = BR14	31:16	Destination Height (in scan lines):
	15:00	Destination Width (in bytes):
3 = BR09	31:00	**Destination Address:** Address of the first byte to be written
	31:14	Reserved.
4 = BR11	15:00	**Source Pitch:** (double word aligned and signed)
5 = BR12	31:00	**Source Address:** Address of the first byte to be read.

1.9 2D (X,Y) BLT Instructions

Most BLT instructions (prefixed with “XY_”) use 2D X,Y coordinate specifications vs. lower-level linear addresses. These instructions also support simple 2D clipping against a clip rectangle.

The top and left Clipping coordinates are inclusive. The bottom and right coordinates are exclusive. The BLT Engine performs a trivial reject for all CLIP BLT instructions before performing any accesses.

Negative destination and source coordinates are supported. In the case of negative source coordinates, the destination X1 and Y1 are modified by the absolute value of the negative source coordinate before the destination clip checking and final drawing coordinates are calculated. The absolute value of the source negative coordinate is added to the corresponding destination coordinate. The BLT engine clipping also checks for (DX2 [or = DX1) or (DY2 [or = DY1) after this calculation and if true, then the BLT is totally rejected.



DX1, DY1, CX1, and CY1 are inclusive, while DX2, DY2, CX2, and CY2 are exclusive.

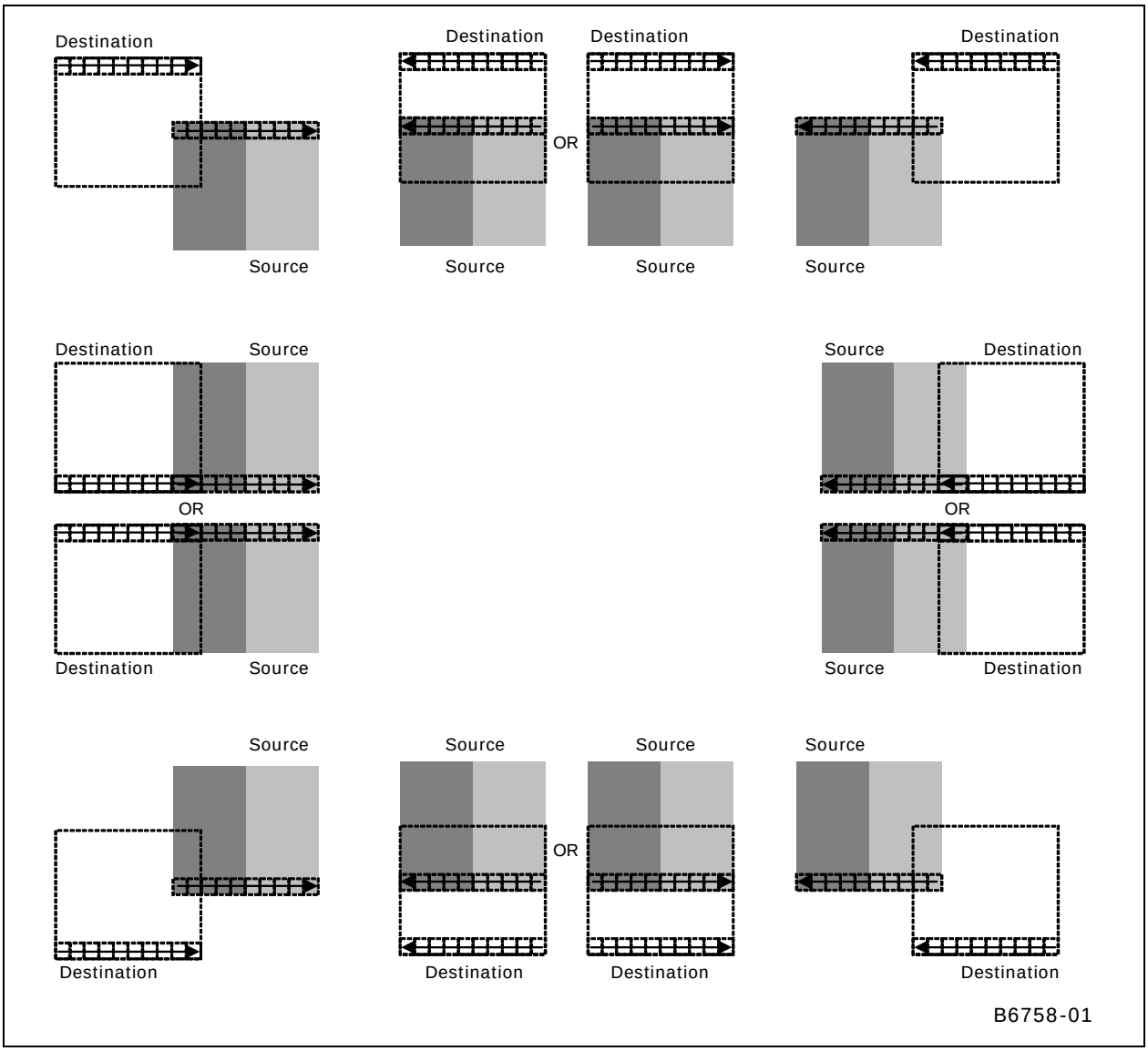
Destination pixel address = (Destination Base Address + (Destination Y coordinate * Destination pitch) + (Destination X coordinate * bytes per pixel)).

Source pixel address = (Source Base Address + (Source Y coordinate * Source pitch) + (Source X coordinate * bytes per pixel)).

Since there is 1 set of Clip Rectangle registers, the Interrupt Ring BLT commands either MUST NEVER enable clipping with these command and never use the XY_Pixel_BLT, XY_Scanline_BLT, nor XY_Text_BLT commands or it must use context switching. The Interrupt rings can also use the non-clipped, linear address commands specified before this section.

The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is **less than** Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and

Source Y1 is **less than** Destination Y1, then the scan line accesses are performed backwards.



1.9.1 XY_SETUP_BLT

This setup instruction supplies common setup information including clipping coordinates used by the XY commands: XY_PIXEL_BLT, XY_SCANLINE_BLT, XY_TEXT_BLT, and XY_TEXT_BLT_IMMEDIATE.

These are the only instructions that require that state be saved between instructions other than the Clipping parameters. There are 5 dedicated registers to contain the state for the 3 setup BLT instructions (XY_SETUP_BLT, XY_SETUP_MONO_PATTERN_SL_BLT, and XY_SETUP_CLIP_BLT. All other BLTs use a temporary version of these. The 5 double word registers are: DW1 (Setup Control), DW6 (Setup

Foreground color), DW5 (Setup Background color), DW7 (Setup Pattern address), and DW4 (Setup Destination Base Address).

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	Instruction Target (Opcode): 01h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:12	Reserved.
	11	Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10: 08	Reserved
	07:00	Dword Length: 06h
1 = BR01	31	Reserved.
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29	**Mono Source Transparency Mode:** (1 = transparency enabled; 0 = use background)
	28:26	Reserved.
	25:24	Color Depth: All 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	Raster Operation:
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement (Negative Pitch Not allowed for Pixel nor Text) For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 = BR24	31:16	**ClipRect Y1 Coordinate (Top):** (30:16 = 15 bit positive number)
	15:00	**ClipRect X1 Coordinate (Left):** (14:00 = 15 bit positive number)
3 = BR25	31:16	**ClipRect Y2 Coordinate (Bottom):** (30:16 = 15 bit positive number)
	15:00	**ClipRect X2 Coordinate (Right):** (14:00 = 15 bit positive number)
4 = BR09	31:00	**Setup Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR05	31:00	**Setup Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0] All
6 = BR06	31:00	**Setup Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0] (SLB & TB only)
7 = BR07	31:00	Setup Pattern Base Address for Color Pattern: (26:06 are implemented) (SLB only) (Note no NPO2 change here). The pattern data must be located in linear memory.

1.9.2 XY_SETUP_MONO_PATTERN_SL_BLT

This setup instruction supplies common setup information including clipping coordinates used exclusively with the following instruction: XY_SCANLINE_BLT (SLB) - 1 scan line of monochrome pattern and destination are the only operands allowed.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	Instruction Target (Opcode): 11h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:12	Reserved.
	11	Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (pre-DevSNB : Tile-X only. DevSNB+ : Tile-X or Tile-Y)
	10: 08	Reserved
	07:00	Dword Length: 07h
1 = BR01	31	**Solid Pattern Select:** (1 = solid pattern; 0 = no solid pattern) - (SLB & Pixel only)
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29	Reserved.
	28	**Mono Pattern Transparency Mode:** (1 = transparency enabled; 0 = use background)
	27:26	Reserved.
	25:24	Color Depth: 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	Raster Operation:
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement (Negative Pitch Not allowed for Pixel nor Text) For Tiled surfaces (bit_11 enabled) this this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 = BR24	31:16	**ClipRect Y1 Coordinate (Top):** (30:16 = 15 bit positive number)
	15:00	**ClipRect X1 Coordinate (Left):** (14:00 = 15 bit positive number)
3 = BR25	31:16	**ClipRect Y2 Coordinate (Bottom):** (30:16 = 15 bit positive number)
	15:00	**ClipRect X2 Coordinate (Right):** (14:00 = 15 bit positive number)
4 = BR09	31:00	**Setup Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR05	31:00	**Setup Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]

DWord	Bit	Description
6 = BR06	31:00	**Setup Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 = BR20	31:00	DW0 (least significant) for a Monochrome Pattern:
8 = BR21	31:00	DW1 (most significant) for a Monochrome Pattern:

1.9.3 XY_SETUP_CLIP_ BLT

This command is used to only change the clip coordinate registers. These are the same clipping registers as the Setup clipping registers above.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	Instruction Target (Opcode): 03h
	21:12	Reserved.
	11	Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10: 08	Reserved
	07:00	Dword Length: 01h
1 = BR24	31:16	**ClipRect Y1 Coordinate (Top):** (30:16 = 15 bit positive number)
	15:00	**ClipRect X1 Coordinate (Left):** (14:00 = 15 bit positive number)
2 = BR25	31:16	**ClipRect Y2 Coordinate (Bottom):** (30:16 = 15 bit positive number)
	15:00	**ClipRect X2 Coordinate (Right):** (14:00 = 15 bit positive number)

1.9.4 XY_PIXEL_BLT

The Destination X coordinate and Destination Y coordinate is compared with the ClipRect registers. If it is within all 4 comparisons, then the pixel supplied in the XY_SETUP_BLT instruction is written with the raster operation to (Destination Y Address + (Destination Y coordinate * Destination pitch) + (Destination X coordinate * bytes per pixel)).

ROP field must specify pattern or fill with 0's or 1's. There is no source operand.

Negative Stride (= Pitch) specified in the Setup command is Not Allowed

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	Instruction Target (Opcode): 24h
	21:12	Reserved.
	11	Tiling Enable:

DWord	Bit	Description
		0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10: 08	Reserved
	07:00	Dword Length : 00h
1 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)

1.9.5 XY_SCANLINES_BLT

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

Solid pattern should use the XY_SETUP_MONO_PATTERN_SL_BLT instruction.

ROP field must specify pattern or fill with 0's or 1's. There is no source operand.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	Instruction Target (Opcode): 25h
	21:15	Reserved.
	14:12	**Pattern Horizontal Seed:** (pixel of the scan line to start on corresponding to DST X=0)
	11	Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10:08	**Pattern Vertical Seed:** (scan line of the 8x8 pattern to start on corresponding to DST Y=0)
	07:00	Dword Length: 01h
1 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)
2 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)

1.9.6 XY_TEXT_BLT

All source scan lines and pixels that fall within the ClipRect Y and X coordinates are written. The source address corresponds to Destination X1 and Y1 coordinate.

Text is either bit or byte packed. Bit packed means that the next scan line starts 1 pixel after the end of the current scan line with no bit padding. Byte packed means that the next scan line starts on the first bit of the next byte boundary after the last bit of the current line.

Source expansion color registers are always in the SETUP_BLT.

Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	Instruction Target (Opcode): 26h
	21:17	Reserved.
	16	**Bit (0) / Byte (1) packed:** Byte packed is for the NT driver
	15:12	Reserved.
	11	Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10: 08	Reserved
	07:00	Dword Length: 02h
1 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)
2 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)
3 = BR12	31:00	**Source Address:** (address of the first byte on scan line corresponding to Dst X1,Y1) (Note no NPO2 change here)

1.9.7 XY_TEXT_IMMEDIATE_BLT

This instruction allows the Driver to send data through the instruction stream that eliminates the read latency of reading a source from memory. If an operand is in system cacheable memory and either small or only accessed once, it can be copied directly to the instruction stream versus to graphics accessible memory.

The IMMEDIATE_BLT data MUST transfer an even number of doublewords. The BLT engine will hang if it does not get an even number of doublewords.

All source scan lines and pixels that fall within the ClipRect X and Y coordinates are written. The source data corresponds to Destination X1 and Y1 coordinate.

Source expansion color registers are always in the SETUP_BLT.

Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h – 2D Processor
	28:22	Instruction Target (Opcode): 31h
	21:17	Reserved.
	16	**Bit (0) / Byte (1) packed:** Byte packed is for the NT driver
	15:12	Reserved.
	11	Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10: 08	Reserved
	07:00	**Dword Length :** 01+ DWL = (Number of Immediate double words)h
1 = BR22	31:16	Destination Y1 Coordinate (Top): (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)
2 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)
3	31:00	Immediate Data DW 0:
4	31:00	Immediate Data DW 1:
5 thru DWL+3	S	Immediate Data DWs 2 through DWORD_LENGTH (DWL):

1.9.8 XY_COLOR_BLT

COLOR_BLT is the simplest BLT operation. It performs a color fill to the destination (with a possible ROP). The only operand is the destination operand which is written dependent on the raster operation. The solid pattern color is stored in the pattern background register.

This instruction is optimized to run at the maximum memory write bandwidth.

The typical (and fastest) Raster operation code = F0 which performs a copy of the pattern background register to the destination.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	Instruction Target (Opcode): 50h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:12	Reserved.
	11	Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10: 08	Reserved
	07:00	Dword Length: 04h
1 = BR13	31	Reserved.
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29:26	Reserved.
	25:24	Color Depth: 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	Raster Operation:
	15:00	Destination Pitch in DWords: [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)
3 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR16	31:00	**Solid Pattern Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]

1.9.9 XY_PAT_BLT

PAT_BLT is used when there is no source and the color pattern is not trivial (is not a solid color only).

If clipping is enabled, all scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	Instruction Target (Opcode): 51h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:15	Reserved.
	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)
	11	Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10:08	**Pattern Vertical Seed: (**Starting Scan line of the 8x8 pattern corresponding to DST Y=0)
	07:00	Dword Length: 04h
1 = BR13	31	Reserved.
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29:26	Reserved.
	25:24	Color Depth: 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	Raster Operation:
	15:00	Destination Pitch in DWords: [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	Destination Y1 Coordinate (Top): (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)
3 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR15	31:00	**Pattern Base Address:** (28:06 are implemented) (Note no NPO2 change here) . The pattern data must be located in linear memory.

1.9.10 XY_PAT_CHROMA_BLT

PAT_BLT is used when there is no source and the color pattern is not trivial (is not a solid color only).

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	Instruction Target (Opcode): 76h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:17	**Transparency Range Mode:** (chroma-key) – Dst Chroma-key modes ONLY (SRC ILLEGAL)
	16:15	Reserved.
	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)
	11	Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10:08	**Pattern Vertical Seed: (**Starting Scan line of the 8x8 pattern corresponding to DST Y=0)
	07:00	Dword Length: 06h
1 = BR13	31	Reserved.
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29:26	Reserved.
	25:24	Color Depth: 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	Raster Operation:
	15:00	Destination Pitch in DWords: [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Yand can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	Destination Y1 Coordinate (Top): (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)
3 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)

DWord	Bit	Description
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR15	31:00	**Pattern Base Address:** (26:06 are used, other bits are ignored) (Note no NPO2 change here). The pattern data must be located in linear memory.
6 = BR18	31:00	**Transparency Color Low:** (Chroma-key Low = Pixel Greater or Equal)
7 = BR19	31:00	**Transparency Color High:** (Chroma-key High = Pixel Less or Equal)

1.9.11 XY_PAT_BLT_IMMEDIATE

PAT_BLT_IMMEDIATE is used when there is no source and the color pattern is not trivial (is not a solid color only) and the pattern is pulled through the command stream. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64DWs) for 8, 16, and 32 bpp color patterns.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	Instruction Target (Opcode): 72h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:15	Reserved.
	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)
	11	Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10:08	**Pattern Vertical Seed:** (starting scan line of the 8x8 pattern corresponding to DST Y=0)
	07:00	**Dword Length:** 03+ DWL = (Number of Immediate double)h
1 = BR13	31	Reserved
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29:26	Reserved.
	25:24	Color Depth: 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color

DWord	Bit	Description
	23:16	Raster Operation:
	15:00	Destination Pitch in DWords: [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)
3 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5	31:00	Immediate Data DW 0:
6	31:00	Immediate Data DW 1:
7 thru DWL+3	S	Immediate Data DWs 2 through DWORD_LENGTH (DWL):

1.9.12 XY_PAT_CHROMA_BLT_IMMEDIATE

PAT_BLT_IMMEDIATE is used when there is no source and the color pattern is not trivial (is not a solid color only) and the pattern is pulled through the command stream. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64DWs) for 8, 16, and 32 bpp color patterns.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	Instruction Target (Opcode): 77h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:17	**Transparency Range Mode:** (chroma-key) – Dst Chroma-key modes ONLY (SRC ILLEGAL)
	16:15	Reserved.
	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)
	11	Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10:08	**Pattern Vertical Seed:** (starting scan line of the 8x8 pattern corresponding to DST Y=0)

DWord	Bit	Description
	07:00	**Dword Length:** 05+ DWL = (Number of Immediate double)h
1 = BR13	31	Reserved.
	30	**Clipping Enable** (1 = enabled; 0 = disabled)
	29:26	Reserved.
	25:24	Color Depth: 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	Raster Operation
	15:00	Destination Pitch in DWords: [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	Destination Y1 Coordinate (Top): (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)
3 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR18	31:00	**Transparency Color Low:** (Chroma-key Low = Pixel Greater or Equal)
6 = BR19	31:00	**Transparency Color High:** (Chroma-key High = Pixel Less or Equal)
7	31:00	Immediate Data DW 0:
8	31:00	Immediate Data DW 1:
9 thru DWL+3	S	Immediate Data DWs 2 through DWORD_LENGTH (DWL):

1.9.13 XY_MONO_PAT_BLT

MONO_PAT_BLT is used when we have no source and the monochrome pattern is not trivial (is not a solid color only). The monochrome pattern is loaded from the instruction stream.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the pattern is 0. When the pattern bit is 1, then the pattern foreground color is used in the ROP operation.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	Instruction Target (Opcode) : 52h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:15	Reserved.
	14:12	**Pattern Horizontal Seed:** (pixel of the scan line to start on corresponding to DST X=0)
	11	Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10:08	**Pattern Vertical Seed:** (starting scan line of the 8x8 pattern corresponding to DST Y=0)
	07:00	Dword Length: 07h
1 = BR13	31	Reserved.
	30	**Clipping Enable** (1 = enabled; 0 = disabled)
	29	Reserved.
	28	**Mono Pattern Transparency Mode:** (1 = transparency enabled; 0 = use background)
	27:26	Reserved.
	25:24	Color Depth: 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	Raster Operation:

DWord	Bit	Description
	15:00	Destination Pitch in DWords: [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Yand can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	Destination Y1 Coordinate (Top): 31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)
3 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR16	31:00	**Pattern Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
6 = BR17	31:00	**Pattern Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 = BR20	31:00	**Pattern Data 0:** (least significant DW)
8 = BR21	31:00	**Pattern Data 1:** (most significant DW)

1.9.14 XY_MONO_PAT_FIXED_BLT

MONO_PAT_FIXED_BLT is used when we have no source and the monochrome pattern is not trivial (is not a solid color only). The monochrome pattern is one of 10 fixed patterns described below. The pattern seeds can still be used with the fixed patterns, creating even more fixed patterns. This eliminates 2 doublewords compared to the XY_MONO_PAT_BLT command packet.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the pattern is 0. When the pattern bit is 1, then the pattern foreground color is used in the ROP operation.

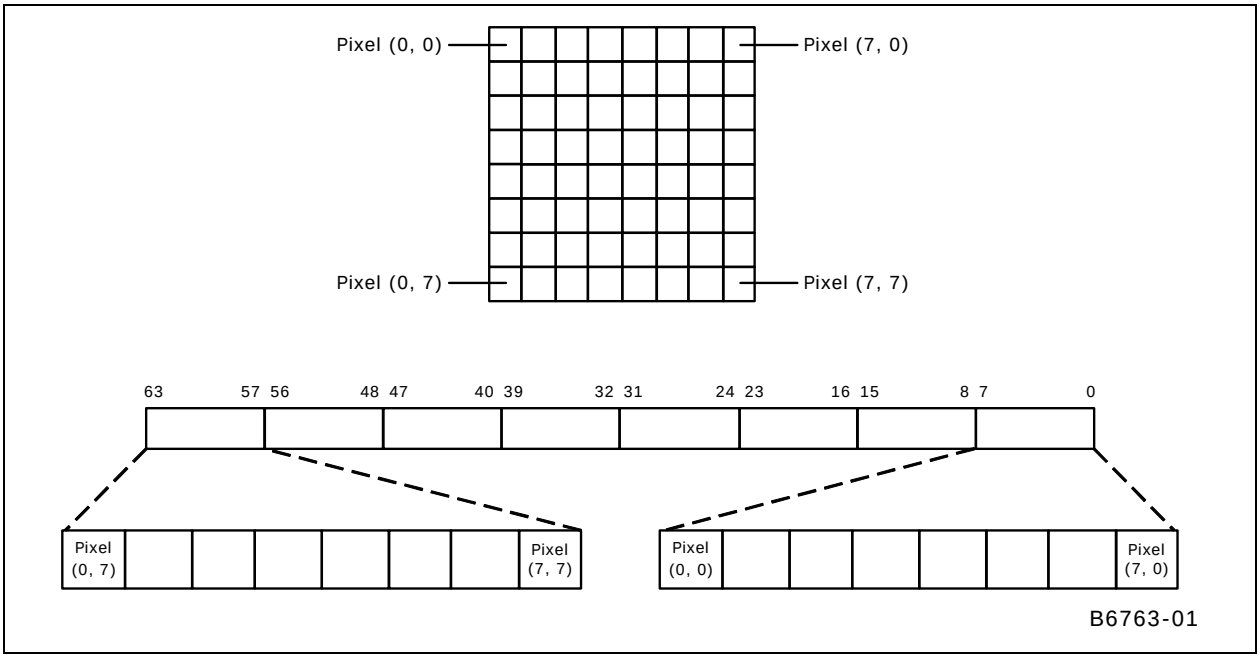
DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	Instruction Target (Opcode): 59h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19	Reserved.
	18:15	Fixed Pattern: 0000 HS_HORIZONTAL 0001 HS_VERTICAL

DWord	Bit	Description
		0010 HS_FDIAGONAL 0011 HS_BDIAGONAL 0100 HS_CROSS 0101 HS_DIAGCROSS 0110 Reserved 0111 Reserved 1000 Screen Door 1001 SD Wide 1010 Walking Bit (one) 1011 Walking Zero 1100 Reserved 1101 Reserved 1110 Reserved
	14:12	**Pattern Horizontal Seed:** (pixel of the scan line to start on corresponding to DST X=0)
	11	Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10:08	**Pattern Vertical Seed:** (starting scan line of the 8x8 pattern corresponding to DST Y=0)
	07:00	Dword Length: 05h
1 = BR13	31	Reserved.
	30	**Clipping Enable** (1 = enabled; 0 = disabled)
	29	Reserved.
	28	**Mono Pattern Transparency Mode:** (1 = transparency enabled; 0 = use background)
	27	**Bit Mask Enable:** (1 = use bit mask register for bit writes; 0 = disabled)
	27:26	Reserved.
	25:24	Color Depth: 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	Raster Operation:
	15:00	Destination Pitch in DWords: [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)

DWord	Bit	Description
3 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR16	31:00	**Pattern Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
6 = BR17	31:00	**Pattern Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]

1.9.14.1 Monochrome Pattern Memory Format

The monochrome pattern is made of 8 bytes that correspond to the 8 pixels per scan line and 8 scan lines. Byte 0 corresponds to scan line 0, byte 1 corresponds to scan line 1,…, and byte 7 corresponds to scan line 7. The bits within each byte are transposed. Pixel 0 is bit 7, pixel 1 is bit 6,…, pixel 7 is bit 0. The diagram below illustrates the byte and bit relationship to the pixels of the pattern.



1.9.14.2 HS_HORIZONTAL 0

Bit 7 0

0,0 7,0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

1.9.14.3 HS_VERTICAL 1

Bit 7 0

0,0 7,0

0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0

1.9.14.4 HS_FDIAGONAL 2

Bit 7 0

0,0 7,0

1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

1.9.14.5 HS_BDIAGONAL 3

Bit 7 0

0,0 7,0

0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0

1.9.14.6 HS_CROSS 4

Bit 7 0

0,0 7,0

0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
1	1	1	1	1	1	1	1
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0

1.9.14.7 HS_DIAGCROSS 5

Bit 7 0

0,0 7,0

1	0	0	0	0	0	0	1
0	1	0	0	0	0	1	0
0	0	1	0	0	1	0	0
0	0	0	1	1	0	0	0
0	0	0	1	1	0	0	0
0	0	1	0	0	1	0	0
0	1	0	0	0	0	1	0
1	0	0	0	0	0	0	1

1.9.14.8 Screen Door 8

Bit 7 0

0,0 7,0

0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0

1.9.14.9 SD Wide 9

Bit 7 0

0,0 7,0

1	1	0	0	1	1	0	0
0	0	1	1	0	0	1	1
1	1	0	0	1	1	0	0
0	0	1	1	0	0	1	1
1	1	0	0	1	1	0	0
0	0	1	1	0	0	1	1
1	1	0	0	1	1	0	0
0	0	1	1	0	0	1	1

1.9.14.10 Walking Bit (One) A

Bit 7 0

0,0 7,0

1	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	0
0	0	0	1	0	0	0	1
1	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	0
0	0	0	1	0	0	0	1

1.9.14.11 Walking Zero B

Bit 7 0

0,0 7,0

0	1	1	1	0	1	1	1
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	0	1	1	1	0
0	1	1	1	0	1	1	1
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	0	1	1	1	0

1.9.15 XY_SRC_COPY_BLT

This BLT instruction performs a color source copy where the only operands involved is a color source and destination of the same bit width.

The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is **less than** Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is **less than** Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.

The ROP value chosen must involve source and no pattern data in the ROP operation.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	Instruction Target (Opcode): 53h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:16	Reserved.
	15	Src Tiling Enable: 0 = Tiling Disabled (Linear) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	14:12	Reserved
	11	Dest Tiling Enable: 0 = Tiling Disabled (Linear) **1 = Tiling enabled (**DevSNB+ : Tile-X or Tile-Y)
	10: 8	Reserved
	7:0	Dword Length: 06h
1 = BR13	31	Reserved.
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29:26	Reserved.
	25:24	Color Depth: 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	Raster Operation

DWord	Bit	Description
	15:00	Destination Pitch in DWords: [15:00] 2's complement For Tiled Dest (bit 11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Yand can be up to 128Kbytes (or 32KDwords).
2 = BR22	31:16	Destination Y1 Coordinate (Top): (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)
3 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Dest Tiling is enabled (Bit 11 enabled), this address is limited to 4Kbytes.
5 = BR26	31:16	**Source Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Source X1 Coordinate (Left):** (15:00 = 16 bit signed number)
6 = BR11	31:16	Reserved
	15:00	Source Pitch (double word aligned) and in DWords: [15:00] 2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Yand can be upto 128Kbytes (or 32KDwords).
7 = BR12	31:00	**Source Base Address:** (base address of the source surface: X=0, Y=0) When Src Tiling is enabled (Bit 15 enabled), this address is limited to 4Kbytes.

1.9.16 XY_SRC_COPY_CHROMA_BLT

This BLT instruction performs a color source copy with chroma-keying where the only operands involved is a color source and destination of the same bit width.

The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is **less than** Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is **less than** Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.

The ROP value chosen must involve source and no pattern data in the ROP operation.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	Instruction Target (Opcode): 73h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:17	Transparency Range Mode: (chroma-key)
	16	Reserved

DWord	Bit	Description
	15	Src Tiling Enable: 0 = Tiling Disabled (Linear) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	14:12	Reserved
	11	Dest Tiling Enable: 0 = Tiling Disabled (Linear) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10: 08	Reserved
	07:00	Dword Length: 08h
1 = BR13	31	Reserved.
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29:26	Reserved.
	25:24	Color Depth: 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	Raster Operation:
	15:00	Destination Pitch in DWords: [15:00] 2's complement For Tiled Dest (bit 11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Yand can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	Destination Y1 Coordinate (Top): (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)
3 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Dest Tiling is enabled (Bit 11 enabled), this address is limited to 4Kbytes.
5 = BR26	31:16	**Source Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Source X1 Coordinate (Left):** (15:00 = 16 bit signed number)
6 = BR11	31:16	Reserved.
	15:00	Source Pitch (double word aligned) and in DWords: [15:00] 2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Yand can be upto 128Kbytes (or 32KDwords).
7 = BR12	31:00	**Source Base Address:** (base address of the source surface: X=0, Y=0) When Src Tiling is enabled (Bit 15 enabled), this address is limited to 4Kbytes.
8 = BR18	31:00	**Transparency Color Low:** (Chroma-key Low = Pixel Greater or Equal)
9 = BR19	31:00	**Transparency Color High:** (Chroma-key High = Pixel Less or Equal)

1.9.17 XY_MONO_SRC_COPY_BLT

This BLT instruction performs a monochrome source copy where the only operands involved is a monochrome source and destination. The source and destination operands cannot overlap therefore the X and Y directions are always forward.

All non-text monochrome sources are word aligned. At the end of a scan line of monochrome source, all bits until the next word boundary are ignored. The monochrome source data bit position field [2:0] indicates the bit position within the first byte of the scan line that should be used as the first source pixel which corresponds to the destination X1 coordinate.

The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation. The ROP value chosen must involve source and no pattern data in the ROP operation.

Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	Instruction Target (Opcode): 54h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:17	Monochrome source data bit position of the first pixel within a byte per scan line.
	16:12	Reserved.
	11	Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10: 08	Reserved
	07:00	Doubleword Length: 06h
1 = BR13	31	Reserved.
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29	**Mono Source Transparency Mode:** (1 = transparency enabled; 0 = use background)
	28:26	Reserved.
	25:24	Color Depth: 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	Raster Operation:
	15:00	Destination Pitch in DWords: [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Yand can be upto 128Kbytes (or 32KDwords).

DWord	Bit	Description
2 = BR22	31:16	Destination Y1 Coordinate (Top): (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)
3 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR12	31:00	**Source Address:** (address corresponding to DST X1,Y1) (Note no NPO2 change here)
6 = BR18	31:00	**Source Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 = BR19	31:00	**Source Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]

1.9.18 XY_MONO_SRC_COPY_ IMMEDIATE_BLT

This instruction allows the Driver to send monochrome data through the instruction stream, eliminating the read latency of the source during command execution.

The IMMEDIATE_BLT data MUST transfer an even number of doublewords and the exact number of quadwords.

All non-text monochrome sources are word aligned. At the end of a scan line of monochrome source, all bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates the bit position within the first byte of the scan line that should be used as the first source pixel which corresponds to the destination X1 coordinate.

The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation. The ROP value chosen must involve source and no pattern data in the ROP operation.

The monochrome source data supplied corresponds to the Destination X1 and Y1 coordinates.

Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	Instruction Target (Opcode): 71h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:17	Monochrome source data bit position of the first pixel within a byte per scan line.
	16:12	Reserved.
	11	Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)

DWord	Bit	Description
	10: 08	Reserved
	07:00	**Dword Length:** 05+ DWL = (Number of Immediate double words)h
1 = BR13	31	Reserved.
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29	**Mono Source Transparency Mode:** (1 = transparency enabled; 0 = use background)
	28:26	Reserved.
	25:24	Color Depth: 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	Raster Operation:
	15:00	Destination Pitch in DWords: [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	Destination Y1 Coordinate (Top): (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)
3 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR18	31:00	**Source Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
6 = BR19	31:00	**Source Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7	31:00	Immediate Data DW 0:
8	31:00	Immediate Data DW 1:
9 thru DWL+4	S	Immediate Data DWs 2 through DWORD_LENGTH (DWL):

1.9.19 XY_FULL_BLT

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source and pattern operands are the same bit width as the destination operand.

The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is **less than** Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is **less than** Destination Y1,

then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	Instruction Target (Opcode): 55h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:16	Reserved.
	15	Src Tiling Enable: 0 = Tiling Disabled (Linear) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)
	11	Dest Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10:08	**Pattern Vertical Seed:** (starting scan line of the 8x8 pattern corresponding to DST Y=0)
	07:00	Doubleword Length: 07h
1 = BR13	31	Reserved.
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29:26	Reserved.
	25:24	Color Depth: 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	Raster Operation:
	15:00	Destination Pitch in DWords: [15:00] 2's complement For Tiled Dest (bit 11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Yand can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)
3 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)

DWord	Bit	Description
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Dest Tiling is enabled (Bit 11 enabled), this address is limited to 4Kbytes.
5 = BR11	31:16	Reserved.
	15:00	Source Pitch (double word aligned and signed) and in DWords: [15:00] 2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
6 = BR26	31:16	**Source Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Source X1 Coordinate (Left):** (15:00 = 16 bit signed number)
7 = BR12	31:00	**Source Base Address:** (base address of the source surface: X=0, Y=0) When Src Tiling is enabled (Bit 15 enabled), this address is limited to 4Kbytes.
8 = BR15	31:00	**Pattern Base Address:** (28:06 are implemented) (Note no NPO2 change here). The pattern data must be located in linear memory.

1.9.20 XY_FULL_IMMEDIATE_PATTERN_BLT

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source and immediate pattern operands are the same bit width as the destination operand. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64 DWs) for 8, 16, and 32 bpp color patterns.

The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is **less than** Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is **less than** Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 = BR00	31:29	**Client**: 02h - 2D Processor
	28:22	Instruction Target (Opcode): 74h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:16	Reserved.

DWord	Bit	Description
	15	Src Tiling Enable: 0 = Tiling Disabled (Linear) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	14:12	**Pattern Horizontal Seed:** (pixel of the scan line to start on corresponding to DST X=0)
	11	Dest Tiling Enable: 0 = Tiling Disabled (Linear) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10:8	**Pattern Vertical Seed:** (starting scan line of the 8x8 pattern corresponding to DST Y=0)
	7:0	**Doubleword Length:** 06+ DWL = (Number of Immediate double words)h
1 = BR13	31	Reserved.
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29:26	Reserved.
	25:24	Color Depth: 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	Raster Operation:
	15:00	Destination Pitch in DWords: [15:00] 2's complement For Tiled Dest (bit 11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)
3 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Dest Tiling is enabled (Bit 11 enabled), this address is limited to 4Kbytes.
5 = BR11	31:16	Reserved.
	15:00	Source Pitch (double word aligned and signed) and in DWords: [15:00] 2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
6 = BR26	31:16	**Source Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Source X1 Coordinate (Left):** (15:00 = 16 bit signed number)
7 = BR12	31:00	**Source Base Address:** (base address of the source surface: X=0, Y=0) When Src Tiling is enabled (Bit 15 enabled), this address is limited to 4Kbytes.
8	31:00	Immediate Data DW 0:
9	31:00	Immediate Data DW 1:

DWord	Bit	Description
A thru DWL+4	S	Immediate Data DWs 2 through DWORD_LENGTH (DWL):

1.9.21 XY_FULL_MONO_SRC_BLT

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source operand is monochrome and the pattern operand is the same bit width as the destination.

The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation.

All non-text and non-immediate monochrome sources are word aligned. At the end of a scan line the monochrome source, the remaining bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates which bit position within the first byte should be used as the first source pixel which corresponds to the Destination X1 coordinate.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

Negative Stride (= Pitch) is NOT ALLOWED

DWord	Bit	Description
0 = BR00	31:29	**Client**: 02h - 2D Processor
	28:22	Instruction Target (Opcode): 56h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:17	Monochrome source data bit position of the first pixel within a byte per scan line.
	16:15	Reserved.
	14:12	**Pattern Horizontal Seed:** (pixel of the scan line to start on corresponding to DST X=0)
	11	Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10:08	**Pattern Vertical Seed:** (starting address of the 8x8 pattern corresponding to DST Y=0)
	07:00	Doubleword Length : 07h
1 = BR13	31	Reserved.
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29	**Mono Source Transparency Mode:** (1 = transparency enabled; 0 = use background)

DWord	Bit	Description
	28:27	Reserved.
	26	Reserved.
	25:24	Color Depth: 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	Raster operation:
	15:00	Destination Pitch in DWords: [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)
3 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR12	31:00	**Mono Source Address:** (address corresponds to DST X1, Y1) (Note no NPO2 change here)
6 = BR18	31:00	**Source Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 = BR19	31:00	**Source Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
8 = BR15	31:00	**Pattern Base Address:** (28:06 are implemented **)** (Note no NPO2 change here). The pattern data must be located in linear memory.

1.9.22 XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source operand is a monochrome and the immediate pattern operand is the same bit width as the destination. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64DWs) for 8, 16, and 32 bpp color patterns.

The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation.

All non-text monochrome sources are word aligned. At the end of a scan line the monochrome source, the remaining bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates which bit position within the first byte should be used as the first source pixel which corresponds to the destination X1 coordinate.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 = BR00	31:29	**Client**: 02h - 2D Processor
	28:22	Instruction Target (Opcode): 75h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:17	Monochrome source data bit position of the first pixel within a byte per scan line.
	16:15	Reserved.
	14:12	**Pattern Horizontal Seed:** (pixel of the scan line to start on corresponding to DST X=0)
	11	Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10:08	**Pattern Vertical Seed:** (starting address of the 8x8 pattern corresponding to DST Y=0)
	07:00	**Doubleword Length :** 06+ DWL = (Number of Immediate double words)h
1 = BR13	31	Reserved.
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29	**Mono Source Transparency Mode:** (1 = transparency enabled; 0 = use background)
	28:26	Reserved.
	25:24	Color Depth: 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	Raster operation:
	15:00	Destination Pitch in DWords: [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)
3 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.

DWord	Bit	Description
5 = BR12	31:00	**Mono Source Address:** (address corresponds to DST X1, Y1) (Note no NPO2 change here)
6 = BR18	31:00	**Source Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 = BR19	31:00	**Source Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
8	31:00	Immediate Data DW 0:
9	31:00	Immediate Data DW 1:
A thru DWL+4	S	Immediate Data DWs 2 through DWORD_LENGTH (DWL):

1.9.23 XY_FULL_MONO_PATTERN_BLT

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The pattern operand is monochrome and the source operand is the same bit width as the destination operand.

The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is **less than** Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is **less than** Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.

The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the pattern foreground color is used in the ROP operation.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

Setting both Solid Pattern Select =1 & Mono Pattern Transparency = 1 is mutually exclusive. The device implementation results in NO PIXELs DRAWN.

DWord	Bit	Description
0 = BR00	31:29	**Client**: 02h - 2D Processor
	28:22	Instruction Target (Opcode): 57h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:16	Reserved.

DWord	Bit	Description
	15	Src Tiling Enable: 0 = Tiling Disabled (Linear) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	14:12	**Pattern Horizontal Seed:** (pixel of the scan line to start on corresponding to DST X=0)
	11	Dest Tiling Enable: 0 = Tiling Disabled (Linear) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10:08	**Pattern Vertical Seed:** (starting scan line of the 8x8 pattern corresponding to DST Y=0)
	07:00	Dword Length : 0Ah
1 = BR13	31	**Solid Pattern Select:** (1 = solid pattern; 0 = no solid pattern)
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29	Reserved.
	28:27	**Mono Pattern Transparency Mode:** (1 = transparency enabled; 0 = use background)
	26	Reserved.
	25:24	Color Depth: 00 = 8 bit color 01 = 16 bit color 10 = 16 bit color (1555) 11 = 32 bit color (565)
	23:16	Raster Operation:
	15:00	Destination Pitch in DWords: [15:00] 2's complement For Tiled Dest (bit 11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)
3 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Dest Tiling is enabled (Bit 11 enabled), this address is limited to 4Kbytes.
5 = BR11	31:16	Reserved.
	15:00	Source Pitch (double word aligned and signed) and in DWords: [15:00] 2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
6 = BR26	31:16	**Source Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Source X1 Coordinate (Left):** (15:00 = 16 bit signed number)
7 = BR12	31:00	**Source Base Address:** (base address of the source surface: X=0, Y=0) When Src Tiling is enabled (Bit 15 enabled), this address is limited to 4Kbytes.

DWord	Bit	Description
8 = BR16	31:00	**Pattern Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
9 = BR17	31:00	**Pattern Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
A = BR20	31:00	**Pattern Data 0:** (least significant DW)
B = BR21	31:00	**Pattern Data 1:** (most significant DW)

1.9.24 XY_FULL_MONO_PATTERN_MONO_SRC_BLT

The full BLT provides the ability to specify all 3 operands: destination, source, and pattern. The pattern and source operands are monochrome.

The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation.

All non-text monochrome sources are word aligned. At the end of a scan line the monochrome source, the remaining bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates which bit position within the first byte should be used as the first source pixel which corresponds to the destination X1 coordinate.

The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the pattern is 0. When the source bit is 1, then the pattern foreground color is used in the ROP operation. The monochrome source transparency mode works identical to the pattern transparency mode.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

Setting both Solid Pattern Select =1 & Mono Pattern Transparency = 1 is mutually exclusive. The device implementation results in NO PIXELs DRAWN.

Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 = BR00	31:29	**Client**: 02h - 2D Processor
	28:22	Instruction Target (Opcode): 58h
	21:20	**32 bpp byte mask:** (21 = 1 = write alpha channel; 20 = 1 = write RGB channels)
	19:17	Monochrome source data bit position of the first pixel within a byte per scan line.
	16:15	Reserved.
	14:12	**Pattern Horizontal Seed:** (pixel of the scan line to start on corresponding to DST X = 0)

DWord	Bit	Description
	11	Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
	10:08	**Pattern Vertical Seed:** (starting scan line of the 8x8 pattern corresponding to DST Y = 0)
	07:00	Doubleword Length : 0Ah
1 = BR13	31	**Solid Pattern Select:** (1 = solid pattern; 0 = no solid pattern)
	30	**Clipping Enable** (1 = enabled; 0 = disabled)
	29	**Mono Source Transparency Mode:** (1 = transparency enabled; 0 = use background)
	28	**Mono Pattern Transparency Mode:** (1 = transparency enabled; 0 = use background)
	27:26	Reserved.
	25:24	Color Depth: 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	Raster Operation:
	15:00	Destination Pitch in DWords: [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	Destination X1 Coordinate (Left): (15:00 = 16 bit signed number)
3 = BR23	31:16	Destination Y2 Coordinate (Bottom): (31:16 = 16 bit signed number)
	15:00	Destination X2 Coordinate (Right): (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR12	31:00	**Source Address:** (address corresponding to Dst X1,Y1) (Note no NPO2 change here)
6 = BR18	31:00	**Source Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 = BR19	31:00	**Source Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
8 = BR16	31:00	**Pattern Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
9 = BR17	31:00	**Pattern Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
A =BR20	31:00	**Pattern Data 0:** (least significant DW)
B =BR21	31:00	**Pattern Data 1:** (most significant DW)

1.10 BLT Engine Instruction Field Definitions

This section describes the BLT Engine instruction fields. These descriptions are in the format of register descriptions. These registers are internal and are not readable. Some of these registers are state that is saved and restored for supporting separate software threads.

1.10.1 BR00—BLT Opcode & Control

Memory Offset Address: none

Default: 0000 0000

Attributes: not accessible

BR00 is the last executed instruction DWord 0. Bits [22:5] are written by every DW0 of every instruction. Bits [31:30] and [4:0] are status bits. Bits [28:27] are written from the DW0 [15:14] of a Setup instruction and Bit 29 is written with a 1 when ever a Setup instruction is written. Bit 29 is a decode of the Setup instruction Opcode.

31	30	29	28 … 24
Rsvd	Clip Inst	Setup Mono Pattern	Instruction Target (Opcode)

23 … 22	21 … 20	19 … 17	16
Instruction Target (Opcode)	32 bpp byte mask	Monochrome Source Start	Bit (0) / Byte (1) Packed

15	14 … 12	11	10 … 8
Rsvd	Pattern Horizontal Seed	Tiling Enable	Transparency Range Mode

7 … 5	4	3	2	1	0
PatternVertical Seed	DST RMW	Color Source	Mono Source	Color Pattern	Mono Pattern

Bit	Descriptions
31	**BLT Engine Busy.** This bit indicates whether the BLT Engine is busy (1) or idle (0). This bit is replicated in the SETUP BLT Opcode & Control register. 1 = Busy 0 = Idle
30	**Setup Instruction Instruction.** The current instruction performs clipping (1).
29	**Setup Monochrome Pattern.** This bit is decoded from the Setup instruction opcode to identify whether a color (0) or monochrome (1) pattern is used with the SCANLINE_BLT instruction. 1 = Monochrome 0 = Color
28:22	**Instruction Target (Opcode).** This is the contents of the Instruction Target field from the last BLT instruction. This field is used by the BLT Engine state machine to identify the BLT instruction it is to perform. The opcode specifies whether the source and pattern operands are color or monochrome.
21:20	**32 bpp byte mask:** 21 = 1 = write alpha channel [31:24]; 20 = 1 = write RGB channels [23:00]. This field is only used for 32bpp.
19:17	**Monochrome Source Start.** This field indicates the starting monochrome pixel bit position within a byte per scan line of the source operand. The monochrome source is word aligned which means that at the end of the scan line all bits should be discarded until the next word boundary.
16	**Bit/Byte Packed .** Byte packed is for the NT driver 0 = Bit 1 = Byte
15	Src Tiling Enable: 0 = Tiling Disabled (Linear) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y)
14:12	**Horizontal Pattern Seed.** This field indicates the pattern pixel position which corresponds to X = 0.
11	Dest Tiling Enable: 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (DevSNB+ : Tile-X or Tile-Y) When set to '1', this means that Blitter is executing in Tiled mode. If '0' it means that Blitter is in Linear mode. Pre-Dev Blitter never executes in Tiled-Y mode, DevSNB+ Blitter supports both Tile-X and Tile-Y modes. On reset, this bit will be '0'. This definition applies to only X,Y Blits. Non-XY blits (COLOR_BLT, SRC_COPY_BLT), will support only linear mode and will not support tiling and for them this bit will remain reserved.

Bit	Descriptions
10:8	**Transparency Range Mode.** These bits control whether or not the byte(s) at the destination corresponding to a given pixel will be conditionally written, and what those conditions are. This feature can make it possible to perform various masking functions in order to selectively write or preserve graphics data already at the destination. XX0 = No color transparency mode enabled. This causes normal operation with regard to writing data to the destination. 001 = **[Source color transparency]** The **Transparency Color Low:** (Pixel Greater or Equal) (source background register) and the **Transparency Color High:** (Pixel Less or Equal) (source foreground register) are compared to the source pixels. The range comparisons are done on each component (R,G,B) and then logically ANDed. If the source pixel components are not within the range defined by the Transparency Color registers, then the byte(s) at the destination corresponding to the current pixel are written with the result of the bit-wise operation. 011 = **[Source and Alpha color transparency]** The **Transparency Color Low:** (Pixel Greater or Equal) (source background register) and the **Transparency Color High:** (Pixel Less or Equal) (source foreground register) are compared to the source pixels. The range comparisons are done on each component (A,R,G,B) and then logically ANDed. If the source pixel components are not within the range defined by the Transparency Color registers, then the byte(s) at the destination corresponding to the current pixel are written with the result of the bit-wise operation. 101 = **[Destination and Alpha color transparency]** The **Transparency Color Low:** (Pixel Greater or Equal) (source background register) and the **Transparency Color High:** (Pixel Less or Equal) (source foreground register) are compared to the destination pixels. The range comparisons are done on each component (A,R,G,B) and then logically ANDed. If the destination pixels are within the range, then the byte(s) at the destination corresponding to the current pixel are written with the result of the bit-wise operation. 111 = **[Destination color transparency]** The **Transparency Color Low:** (Pixel Greater or Equal) (source background register) and the **Transparency Color High:** (Pixel Less or Equal) (source foreground register) are compared to the destination pixels. The range comparisons are done on each component (R,G,B) and then logically ANDed. If the destination pixels are within the range, then the byte(s) at the destination corresponding to the current pixel are written with the result of the bit-wise operation.
7:5	**Pattern Vertical Seed.** This field specifies the pattern scan line which corresponds to Y=0.
4	**Destination Read Modify Write.** This bit is decoded from the last instruction's opcode field and Destination Transparency Mode to identify whether a Destination read is needed.
3	**Color Source.** This bit is decoded from the last instructions opcode field to identify whether a color (1) source is used.
2	**Monochrome Source.** This bit is decoded from the last instructions opcode field to identify whether a monochrome (1) source is used.
1	**Color Pattern.** This bit is decoded from the last instructions opcode field to identify whether a color (1) pattern is used.
0	**Monochrome Pattern.** This bit is decoded from the last instructions opcode field to identify whether a monochrome (1) pattern is used.

1.10.2 BR01—Setup BLT Raster OP, Control, and Destination Offset

Memory Offset Address: none

Default: 0000 xxxx

Attributes: State accessible

BR01 contains the contents of the last Setup instruction DWord 1. It is identical to the BLT Raster OP, Control, and Destination Offset definition, but it is used with the following instructions: PIXEL_BLT, SCANLINE_BLT, and TEXT_BLT.

31	30	29	28	27 26	25 24
Solid Pattern	Clipping Enable	Mono Src Trans	Mono Pat Trans	32 bpp byte mask	Color Depth

23 16
Raster Operation

15 0
Destination Pitch (Offset)

Bit	Descriptions
31	**Solid Pattern Select.** This bit applies only when the pattern data is monochrome. This bit determines whether or not the BLT Engine actually performs read operations from the frame buffer in order to load the pattern data. Use of this feature to prevent these read operations can increase BLT Engine performance, if use of the pattern data is indeed not necessary. The BLT Engine is configured to accept either monochrome or color pattern data via the opcode field. 0 = This causes normal operation with regard to the use of the pattern data. The BLT Engine proceeds with the process of reading the pattern data, and the pattern data is used as the pattern operand for all bit-wise operations. 1 = The BLT Engine forgoes the process of reading the pattern data, the presumption is made that all of the bits of the pattern data are set to 0, and the pattern operand for all bit-wise operations is forced to the background color specified in the Color Expansion Background Color Register.
30	**Clipping Enabled**: 1 = Enabled; 0 = Disabled

Bit	Descriptions
29	**Monochrome Source Transparency Mode.** This bit applies only when the source data is in monochrome. This bit determines whether or not the byte(s) at the destination corresponding to the pixel to which a given bit of the source data also corresponds will actually be written if that source data bit has the value of 0. This feature can make it possible to use the source as a transparency mask. The BLT Engine is configured to accepted either monochrome or color source data via the opcode field. 0 = This causes normal operation with regard to the use of the source data. Wherever a bit in the source data has the value of 0, the color specified in the background color register is used as the source operand in the bit-wise operation for the pixel corresponding to the source data bit, and the bytes at the destination corresponding to that pixel are written with the result. 1 = Wherever a bit in the source data has the value of 0, the byte(s) at the destination corresponding to the pixel to which the source data bit also corresponds are simply not written, and the data at those byte(s) at the destination are allowed to remain unchanged.
28	**Monochrome Pattern Transparency Mode.** This bit applies only when the pattern data is monochrome. This bit determines whether or not the byte(s) at the destination corresponding to the pixel to which a given bit of the pattern data also corresponds will actually be written if that pattern data bit has the value of 1. This feature can make it possible to use the pattern as a transparency mask. The BLT Engine is configured to accepted either monochrome or color pattern data via the opcode field. 0 = This causes normal operation with regard to the use of the pattern data. Wherever a bit in the pattern data has the value of 0, the color specified in the background color register is used as the pattern operand in the bit-wise operation for the pixel corresponding to the pattern data bit, and the bytes at the destination corresponding to that pixel are written with the result. 1 = Wherever a bit in the pattern data has the value of 0, the byte(s) at the destination corresponding to the pixel to which the pattern data bit also corresponds are simply not written, and the data at those byte(s) at the destination are allowed to remain unchanged.
27:26	**32 bpp byte mask.** 21 = 1 = write alpha channel [31:24]; 20 = 1 = write RGB channels [23:00]. This field is only used for 32bpp.
25:24	Color Depth. 00 = 8 Bit Color Depth 01 = 16 Bit Color Depth 10 = 16 Bit Color Depth 11 = 32 Bit Color Depth
23:16	**Raster Operation Select.** These 8 bits are used to select which one of 256 possible raster operations is to be performed by the BLT Engine. The 8-bit values, and their corresponding raster operations, are intended to correspond to the 256 possible raster operations specified for graphics device drivers The opcode field must indicate a monochrome source if ROP = F0.

Bit	Descriptions
15:0	Destination Pitch (Offset). For non-XY Blits, the signed 16bit field allows for specifying upto ± 32Kbytes signed pitches in bytes (same as before). For X, Y Blits with tiled-X surfaces, the pitch for Destination will be 512Byte aligned and should be programmable upto ± 128Kbytes. For X, Y Blits with tiled-Y surfaces, the pitch for Destination will be 128Byte aligned and should be programmable upto ± 128Kbytes. In this case, this 16bit signed pitch field is used to specify upto ± 32K**DWords**. For X, Y blits with nontiled surfaces (linear surfaces), this 16bit field can be programmed to byte specification of upto ± 32Kbytes (same as before). These 16 bits store the signed memory address offset value by which the destination address originally specified in the Destination Address Register is incremented or decremented as each scan line's worth of destination data is written into the frame buffer by the BLT Engine, so that the destination address will point to the next memory address to which the next scan line's worth of destination data is to be written. If the intended destination of a BLT operation is within on-screen frame buffer memory, this offset is normally set so that each subsequent scan line's worth of destination data lines up vertically with the destination data in the scan line, above. However, if the intended destination of a BLT operation is within off-screen memory, this offset can be set so that each subsequent scan line's worth of destination data is stored at a location immediately after the location where the destination data for the last scan line ended, in order to create a single contiguous block of bytes of destination data at the destination.

1.10.3 BR05—Setup Expansion Background Color

Memory Offset Address: none

Default: None

Attributes: State accessible

31	0
Setup Expansion Background Color Bits [31:0]	

Bit	Descriptions
31:0	**Setup Expansion Background Color Bits [31:0].** These bits provide the one, two, or four bytes worth of color data that select the background color to be used in the color expansion of monochrome pattern or source data for either the SCANLINE_BLT or TEXT_BLT instructions. BR05 is also used as the solid pattern for the PIXEL_BLT instruction. Whether one, two, or three bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

1.10.4 BR06—Setup Expansion Foreground Color

Memory Offset Address: none

Default: None

Attributes: State accessible

31	0
Setup Expansion Foreground Color Bits [31:0]	

Bit	Descriptions
31:24	Reserved.
31:0	**Setup Expansion Foreground Color Bits [31:0].** These bits provide the one, two, or four bytes worth of color data that select the foreground color to be used in the color expansion of monochrome pattern or source data for either the SCANLINE_BLT or TEXT_BLT instructions. Whether one, two, or three bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

1.10.5 BR07—Setup Color Pattern Address

Memory Offset Address: none

Default: None

Attributes: State accessible

31 29	28 16
Reserved	Setup Color Pattern Address Bits [28:16]

15 6	5 0
Setup Color Pattern Address Bits [15:6]	Reserved

Bit	Descriptions
31:29	**Reserved.** The maximum GC graphics address is 512 MBs.
28:6	**Pattern Address.** These 23 bits specify the starting address of the color pattern from the SETUP_BLT instruction. This register works identically to the Pattern Address register, but this version is only used with the SCANLINE_BLT instruction execution. The pattern data must be located in linear memory. The pattern data must be located on a pattern-size boundary. The pattern is always of 8x8 pixels, and therefore, its size is dependent upon its pixel depth. The pixel depth may be 8, 16, or 32 bits per pixel if the pattern is in color (the pixel depth of a color pattern must match the pixel depth to which the graphics system has been set). Monochrome patterns require 8 bytes and is supplied through the instruction. Color patterns of 8, 16, and 32 bits per pixel color depth must start on 64-byte, 128-byte and 256-byte boundaries, respectively.
5:0	**Reserved.** These bits always return 0 when read.

1.10.6 BR09—Destination Address

Memory Offset Address: None

Default: None

Attributes: State accessible

31 29	28 0
Reserved	Destination and Destination Y1 and Y Address Bits [28:0]

Bit	Descriptions
31:29	Reserved.
28:0	**Destination Address Bits.** When tiling is enabled for XY-blits, this base address should be limited to 4KB. Otherwise for XY blits, there is no restriction and it is same as before. These 29 bits specify the starting pixel address of the destination data. This register is also the working destination address register and changes as the BLT Engine performs the accesses. Used as the scan line address (Destination Y Address & Destination Y1 Address) for BLT instructions: PIXEL_BLT, SCANLINE_BLT, and TEXT_BLT. In this case the address points to the first pixel in a scan line and is compared with the ClipRect Y1 & Y2 address registers to determine whether the scan line should be written or not. The Destination Y1 address is the top scan line to be written for text. Note that for non-XY blits (COLOR_BLT, SRC_COPY_BLT), this address points to the first byte to be written. This register is always the last register written for a BLT drawing instruction. Writing BR09 starts the BLT engine execution. ***Note:*** Some instructions affect only one scan line (requiring only one coordinate); other instructions affect multiple scan lines and need both coordinates.

1.10.7 BR11—BLT Source Pitch (Offset)

Memory Offset Address: None

Default: None

Attributes: Not accessible

31 … 16	15 … 0
BLT Engine Status - TBS	Source Pitch (Offset)

Bit	Descriptions
31:16	Reserved
15:0	Source Pitch (Offset) For non-XY Blits with color source operand (SRC_COPY_BLT), the signed 16bit field allows for specifying upto ± 32Kbytes signed pitch in bytes (same as before). For X, Y Blits with tiled-X surfaces, the pitch for Color Source will be 512Byte aligned and should be programmable upto ± 128Kbytes. For X, Y Blits with tiled-Y surfaces, the pitch for Color Source will be 128Byte aligned and should be programmable upto ± 128Kbytes. In this case, this 16bit signed pitch field is used to specify upto ± 32K**DWords**. For X, Y blits with nontiled color source surfaces (linear surfaces), this 16bit field can be programmed to byte specification of upto ± 32Kbytes (same as before). When the color source data is located within the frame buffer or AGP aperture, these signed 16 bits store the memory address offset (pitch) value by which the source address originally specified in the Source Address Register is incremented or decremented as each scan line's worth of source data is read from the frame buffer by the BLT Engine, so that the source address will point to the next memory address from which the next scan line's worth of source data is to be read. Note that if the intended source of a BLT operation is within on-screen frame buffer memory, this offset is normally set to accommodate the fact that each subsequent scan line's worth of source data lines up vertically with the source data in the scan line, above. However, if the intended source of a BLT operation is within off-screen memory, this offset can be set to accommodate a situation in which the source data exists as a single contiguous block of bytes where in each subsequent scan line's worth of source data is stored at a location immediately after the location where the source data for the last scan line ended.

1.10.8 BR12—Source Address

Memory Offset Address: None

Default: None

Attributes: Not accessible

31 29	28 0
Reserved	Source Address Bits [28:0]

Bit	Descriptions
31:29	**Reserved.** The maximum GC Graphics address is 512 MBs.
28:0	**Source Address Bits [28:0].** When tiling is enabled for XY-blits with Color source surfaces, this base address should be limited to 4KB. Otherwise for XY blits, there is no restriction and it is same as before, including for monosource and text blits. Note that for non-XY blit with Color Source (SRC_COPY_BLT), this address points to the first byte to be read. These 29 bits are used to specify the starting pixel address of the color source data. The lower 3 bits are used to indicate the position of the first valid byte within the first Quadword of the source data.

1.10.9 BR13—BLT Raster OP, Control, and Destination Pitch

Memory Offset Address: None

Default: 0000 xxxx

Attributes: Not accessible

31	30	29	28	27 26	25 24
Solid Pattern	Clipping Enable	Mono Src Trans	Mono Pat Trans	32 bpp byte mask	Color Depth

23 16
Raster Operation

15 0
Destination Pitch (Offset)

Bit	Descriptions
31	**Solid Pattern Select.** This bit applies only when the pattern data is monochrome. This bit determines whether or not the BLT Engine actually performs read operations from the frame buffer in order to load the pattern data. Use of this feature to prevent these read operations can increase BLT Engine performance, if use of the pattern data is indeed not necessary. The BLT Engine is configured to accept either monochrome or color pattern data via the opcode field. 0 = This causes normal operation with regard to the use of the pattern data. The BLT Engine proceeds with the process of reading the pattern data, and the pattern data is used as the pattern operand for all bit-wise operations. 1 = The BLT Engine forgoes the process of reading the pattern data, the presumption is made that all of the bits of the pattern data are set to 0, and the pattern operand for all bit-wise operations is forced to the background color specified in the Color Expansion Background Color Register.
30	**Clipping Enabled**: 1 = Enabled; 0 = Disabled
29	**Monochrome Source Transparency Mode.** This bit applies only when the source data is in monochrome. This bit determines whether or not the byte(s) at the destination corresponding to the pixel to which a given bit of the source data also corresponds will actually be written if that source data bit has the value of 0. This feature can make it possible to use the source as a transparency mask. The BLT Engine is configured to accepted either monochrome or color source data via the opcode field. 0 = This causes normal operation with regard to the use of the source data. Wherever a bit in the source data has the value of 0, the color specified in the background color register is used as the source operand in the bit-wise operation for the pixel corresponding to the source data bit, and the bytes at the destination corresponding to that pixel are written with the result. 1 = Where a bit in the source data has the value of 0, the byte(s) at the destination corresponding to the pixel to which the source data bit also corresponds are simply not written, and the data at those byte(s) at the destination are allowed to remain unchanged.
28	**Monochrome Pattern Transparency Mode.** This bit applies only when the pattern data is monochrome. This bit determines whether or not the byte(s) at the destination corresponding to the pixel to which a given bit of the pattern data also corresponds will actually be written if that pattern data bit has the value of 1. This feature can make it possible to use the pattern as a transparency mask. The BLT Engine is configured to accepted either monochrome or color pattern data via the opcode in the Opcode and Control register. 0 = This causes normal operation with regard to the use of the pattern data. Where a bit in the pattern data has the value of 0, the color specified in the background color register is used as the pattern operand in the bit-wise operation for the pixel corresponding to the pattern data bit, and the bytes at the destination corresponding to that pixel are written with the result. 1= Wherever a bit in the pattern data has the value of 0, the byte(s) at the destination corresponding to the pixel to which the pattern data bit also corresponds are simply not written, and the data at those byte(s) at the destination are allowed to remain unchanged.
25:24	Color Depth. 00 = 8 Bit Color Depth 01 = 16 Bit Color Depth 10 = 24 Bit Color Depth 11 = Reserved
23:16	**Raster Operation Select.** These 8 bits are used to select which one of 256 possible raster operations is to be performed by the BLT Engine. The 8-bit values, and their corresponding raster operations, are intended to correspond to the 256 possible raster operations specified for graphics device drivers. The opcode must indicate a monochrome source operand if ROP = F0.

Bit	Descriptions
15:0	**Destination Pitch (Offset).** These 16 bits store the signed memory address offset value by which the destination address originally specified in the Destination Address Register is incremented or decremented as each scan line's worth of destination data is written into the frame buffer by the BLT Engine, so that the destination address will point to the next memory address to which the next scan line's worth of destination data is to be written. If the intended destination of a BLT operation is within on-screen frame buffer memory, this offset is normally set so that each subsequent scan line's worth of destination data lines up vertically with the destination data in the scan line, above. However, if the intended destination of a BLT operation is within off-screen memory, this offset can be set so that each subsequent scan line's worth of destination data is stored at a location immediately after the location where the destination data for the last scan line ended, in order to create a single contiguous block of bytes of destination data at the destination.

1.10.10 BR14—Destination Width and Height

Memory Offset Address: None

Default: None

Attributes: Not accessible

BR14 contains the values for the height and width of the data to be BLT. If these values are not correct, such that the BLT Engine is either expecting data it does not receive or receives data it did not expect, the system can hang.

31 29	28 16
Reserved	Destination Height

15 13	12 0
Reserved	Destination Byte Width

Bit	Descriptions
31:29	Reserved.
28:16	**Destination Height.** These 13 bits specify the height of the destination data in terms of the number of scan lines. This is a working register.
15:13	Reserved.
12:0	**Destination Byte Width.** These 13 bits specify the width of the destination data in terms of the number of bytes per scan line. The number of pixels per scan line into which this value translates depends upon the color depth to which the graphics system has been set.

1.10.11 BR15—Color Pattern Address

Memory Offset Address: None

Default: None

Attributes: Not accessible

31 29	28 16
Reserved	Color Pattern Address Bits [28:16]

15 6	5 0
Color Pattern Address Bits [15:6]	Reserved

.

Bit	Descriptions
31:29	**Reserved.** The maximum GC graphics address is 512 MBs.
28:6	**Color Pattern Address.** There is no change to the Color Pattern address specification due to Non-Power-of-2 change. It remains the same as before. The pattern data must be located in linear memory. These 23 bits specify the starting address of the pattern. The pattern data must be located on a pattern-size boundary. The pattern is always of 8x8 pixels, and therefore, its size is dependent upon its pixel depth. The pixel depth may be 8, 16, or 32 bits per pixel if the pattern is in color (the pixel depth of a color pattern must match the pixel depth to which the graphics system has been set). Monochrome patterns require 8 bytes and are applied through the instruction. Color patterns of 8, 16, and 32 bits per pixel color depth must start on 64-byte, 128-byte and 256-byte boundaries, respectively.
5:0	**Reserved.** These bits always return 0 when read.

1.10.12 BR16—Pattern Expansion Background and Solid Pattern Color

Memory Offset Address: 40040h

Default: None

Attributes: RO; DWord accessible

31	0
Pattern Expansion Background Color Bits [31:0]	

.

Bit	Descriptions
31:0	**Pattern Expansion Background Color Bits [31:0].** These bits provide the one, two, or four bytes worth of color data that select the background color to be used in the color expansion of monochrome pattern data during BLT operations. Whether one, two, or four bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

1.10.13 BR17—Pattern Expansion Foreground Color

Memory Offset Address: None

Default: None

Attributes: Not accessible

31	0
Pattern Expansion Foreground Color Bits [31:0]	

Bit	Descriptions
31:0	**Pattern Expansion Foreground Color Bits [31:0].** These bits provide the one, two, or four bytes worth of color data that select the foreground color to be used in the color expansion of monochrome pattern data during BLT operations. Whether one, two, or four bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

1.10.14 BR18—Source Expansion Background, and Destination Color

Memory Offset Address: None

Default: None

Attributes: Not accessible

31	0
Source Expansion Background Color Bits [31:0]	

Bit	Descriptions
31:0	**Source Expansion Background Color Bits [31:0].** These bits provide the one, two, or four bytes worth of color data that select the background color to be used in the color expansion of monochrome source data during BLT operations. This register is also used to support destination transparency mode and Solid color fill. Whether one, two, three, or four bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

1.10.15 BR19—Source Expansion Foreground Color

Memory Offset Address: None

Default: None

Attributes: Not accessible

31	0
Pattern Expansion Foreground Color Bits [31:0]	

Bit	Descriptions
31:0	**Pattern/Source Expansion Foreground Color Bits [31:0].** These bits provide the one, two, or four bytes worth of color data that select the foreground color to be used in the color expansion of monochrome source data during BLT operations. Whether one, two, or four bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

2. Blitter (Blt) Engine Command Streamer

The Blitter register interface is only implemented on DevSNB+.

A separate pipeline for blitter operations is introduced on DevSNB+. This pipeline has its own command streamer and operates completely independently of the other command streamers. This command streamer supports a separate set of registers starting at offset 20000h.

2.1 Registers for Blitter Engine

2.1.1 Introduction

Each register is at the same offset from 020000h as its primary counterpart is offset from 02000h.

2.1.2 GAB PWR CTX STORAGE REGISTERS

2.1.2.1 GAB_CTL_REG – GAB unit Control Register

GAB_CTL_REG	
Register Type:	MMIO
Address Offset:	24000h
Project:	DevSNB+
Default Value:	FF0000BFh
Access:	R/W
Size (in bits):	32
Trusted Type:	1

Bit	Description
31:9	**Reserved**
8	**Continue after Page Fault**. If set to 1: upon receiving a page fault when requesting an address translation, GAB will set address bit 39 to 1 and continue. If set to 0: GAB will hang on a page fault. Default = b0. Control added in C-step ECO. NOTE TO DRIVER: for gen6 this bit needs to be written to “1”
7:6	**PPGTT BCS TLB LRA MIN: TLB Depth Partitioning Register In PP GTT Mode. Default = b10**
5:4	**GAB write request priority signal value used in GAC arbitration. Default = b11**
3:2	**GAB read only request priority signal value used in GAC arbitration. Default = b11**
1:0	**GAB read request priority signal value used in GAC arbitration. Default = b11**

2.1.3 Virtual Memory Control

2.1.3.1 BCS_HWS_PGA — Hardware Status Page Address Register

BCS_HWS_PGA — Hardware Status Page Address Register	
Register Type:	MMIO_BCS
Address Offset:	24080 for [DevSNB]
Project:	DevSNB
Default Value:	0000_0000h
Access:	R/W
Size (in bits):	32
Trusted Type:	1

This register is used to program the 4 KB-aligned System Memory address of the Hardware Status Page used to report hardware status into (typically cacheable) System Memory. [DevSNB] This address in this register is translated using the Global GTT in memory. The mapping type of the GTT entry determines the snoop nature of the transaction to memory.

Programming Notes
[DevSNB+] If this register is written, a workload must subsequently be dispatched to the blitter command streamer.

Bit	Description
31:12	**Address** Project: DevSNB Security: None Address: GraphicsAddress[31:12] This field is used by SW to specify Bits 31:12 of the 4 KB-aligned System Memory address of the 4 KB page known as the "Hardware Status Page". The Global GTT is used to map this page from the graphics virtual address to physical address **Programming Notes** If the Per-Process Virtual Address Space is set, HW requires that the status page is programmed to allow for the context switch status to be reported
12:1	**Reserved** Project: All Format: MBZ
0	**Translation in Progress** Project: DevSNB Access: RO Format: U1 This field indicates that the translation for the hardware status page from the graphics virtual address to the physical address is pending. Software can use this indicator to prevent updating the status page when there is a pending cycle for translation.

The following table defines the layout of the Hardware Status Page:

DWord Offset	Description
3:0	**Reserved.** Must not be used.
4	**Head Pointer Storage:** The contents of the Ring Buffer Head Pointer register (register DWord 1) are written to this location either as result of an MI_REPORT_HEAD instruction or as the result of an "automatic report" (see RINGBUF registers).
0Fh:05h	**Reserved.** Must not be used.
(3FFh – 010h)	These locations can be used for general purpose via the MI_STORE_DATA_INDEX or MI_STORE_DATA_IMM instructions.

2.1.3.2 BCS_PP_DIR_BASE – Page Directory Base Register

BCS_PP_DIR_BASE – Page Directory Base Register	
Register Type:	MMIO_CS
Address Offset:	22228h
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

This register contains the offset into the GGTT where the (current context's) PPGTT page directory begins. This register is restored with context. The Page Directory Base Address is set by SW only by modifying the value of this register in the context image such that the new value is restored the next time the context runs. A write via MMIO to this register triggers the render pipe to fetch all PDs.

Programming Note: The **MBC Driver Boot Enable** bit in MBCTL register must be set *before* this register is written to upon boot up (including S3 exit)

Bit	Description
30:16	**Page Directory Base Offset** Project: All Default Value: 0h Format: U15 Range [0,GGTT Size in cachelines - 1] Contains the cacheline (64-byte) offset into the GGTT where the page directory begins.
15:1	**Reserved** Project: All Format: MBZ
0	**PD Load Busy** Project: DevSNB+ Format: Valid This is a read-only field that indicates if the page directories are currently being fetched and loaded.

2.1.3.3 BCS_PP_DCLV – PPGTT Directory Cacheline Valid Register

BCS_PP_DCLV – PPGTT Directory Cacheline Valid Register	
Register Type:	MMIO_CS
Address Offset:	22220h
Project:	All
Default Value:	0h
Access:	R/W
Size (in bits):	64

This register controls update of the on-chip PPGTT Directory Cache during a context restore. Bits that are set will trigger the load of the corresponding 16 directory entry group. This register is restored with context (prior to restoring the on-chip directory cache itself). This register is also restored when switching to a context whose LRCA matches the current CCID if the **Force PD Restore** bit is set in the context descriptor.

The context image of this register must be updated and maintained by SW; SW should not normally need to read this register.

This register can also effectively be used to limit the size of a processes' virtual address space. Any access by a process that requires a PD entry in a set that is not enabled in this register will cause a fatal error, and no fetch of the PD entry will be attempted

Bit	Description
63:32	**Reserved** Project: All Format: MBZ
31:0	**PPGTT Directory Cache Restore [1..32] 16 entries** Project: All Format: Array:Enable If set, the [1st..32nd] 16 entries of the directory cache are considered valid and will be brought in on context restore. If clear, these entries are considered invalid and fetch of these entries will not be attempted.

The field below needs to go in some register to enable PPGTT, either in GAB MMIO or BCS MMIO

1	**Per-Process GTT Enable** Project: DevSNB+ Format: Enable If set, PPGTT support in hardware is enabled. This bit must be set if runlist enable is set. Setting this bit also allows support for big pages (32k)

2.1.4 Mode and Misc Ctrl Registers

2.1.4.1 BCS_MI_MODE — Mode Register for Software Interface

Address Offset: 2209Ch–2209Fh

Default Value: 0000 0000h

Access: Read/Write

Size: 32 bits

The MI_MODE register contains information that controls software interface aspects of the command parser.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:16</td><td>Masks: A “1” in a bit in this field allows the modification of the corresponding bit in Bits 15:0</td></tr>
<tr><td>15</td><td>Suspend Flush
Project: All
Mask: MMIO(0x209c)#31

<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>No Delay</td><td>HW will not delay flush, this bit will get cleared by MI_SUSPEND_FLUSH as well</td><td>All</td></tr>
<tr><td>1h</td><td>Delay Flush</td><td>HW will delay the flush because of sync flush or VTD regimes until reset, this bit will get set by MI_SUSPEND_FLUSH as well</td><td>All</td></tr>
</table></td></tr>
<tr><td>14:12</td><td>Reserved Read/Write</td></tr>
<tr><td>11</td><td>Invalidate UHPTR enable: If bit set H/W clears the valid bit of BCS_UHPTR (4134h, bit 0) when current active head pointer is equal to UHPTR.</td></tr>
<tr><td>10</td><td>Reserved Read/Write</td></tr>
<tr><td>9</td><td>Ring Idle (Read Only Status bit)
0 = Parser not Idle
1 = Parser Idle
Writes to this bit are not allowed.</td></tr>
</table>

Bit	Description
8	**Stop Ring** 0 = Normal Operation. 1 = Parser is turned off. Software must set this bit to force the Ring and Command Parser to Idle. Software must read a “1” in Ring Idle bit after setting this bit to ensure that the hardware is idle. ***Software must clear this bit for Ring to resume normal operation.***
7:2	**Reserved** Read/Write
1	**Bypass Fence Write [DevSNB]** 0 = Normal Operation. 1 = Bypass. If set, this bit will bypass all writes during flushes, independent of programming. This includes post-sync op bits, the implicit TLB invalidate write (set in GFX_MODE[13]), and sync flush fences. ***Note this is only intended for work-arounds***
0	**Reserved** Read/Write

2.1.4.2 BCS_NOPID — NOP Identification Register

BCS_NOPID — NOP Identification Register	
Register Type:	MMIO_BCS
Address Offset:	22094h
Project:	All
Default Value:	00000000h
Access:	RO
Size (in bits):	32
Trusted Type:	1
The NOPID register contains the Noop Identification value specified by the last MI_NOOP instruction that enabled this register to be updated.	

Bit	Description
31:22	**Reserved** Project: All Format: MBZ
21:0	Identification Number Project: ll Security: one Default Value: h efaultVaueDesc This field contains the 22-bit Noop Identification value specified by the last MI_NOOP instruction that enabled this field to be updated

2.1.4.3 BCS_INSTPM—Instruction Parser Mode Register

Address Offset: 220C0h–220C3h

Default Value: 0000 0000h

Access: Read/Write

Size: 32 bits

The BCS_INSTPM register is used to control the operation of the BCS Instruction Parser. Certain classes of instructions can be disabled (ignored) – often useful for detecting performance bottlenecks. Also, “Synchronizing Flush” operations can be initiated – useful for ensuring the completion (vs. only parsing) of rendering instructions.

Programming Notes:

- All Reserved bits are implemented.

BCS_INSTPM—Instruction Parser Mode Register

Register Type: MMIO_BCS
Address Offset: 220C0h
Project: DevSNB+
Default Value: 00000000h
Access: R/W
Size (in bits): 32
Trusted Type: 1

Bit	Description
31:16	**Mask Bits** Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
15:10	**Reserved** Project: All Format: MBZ
9	**Reserved**
8:7	**Reserved** Project: All Format: MBZ
6	**Memory Sync Enable** Project: DevSNB+ Format: U1 This set, this bit allows the blitter decode engine to write out the data from the local caches to memory. This bit is not persistent. S/W must define this bit each time a sync flush is requested
5	**Sync Flush Enable** Project: DevSNB+ Format: U1 This field is used to request a Sync Flush operation. The device will automatically clear this bit before completing the operation. See Sync Flush (Programming Environment). Format = Enable (cleared by HW) [DevSNB] When using MI_SUSPEND_FLUSH, this bit cannot be relied on as an indicator of sync flush complete. Instead, driver must wait until head == tail
4:0	**Reserved** Project: All Format: MBZ

2.1.4.4 BCS_EXCC—Execute Condition Code Register

BCS_EXCC—Execute Condition Code Register	
Register Type:	MMIO_VCS
Address Offset:	22028h
Project:	All
Default Value:	00000000h
Access:	R/W,RO
Size (in bits):	32
Trusted Type:	1

This register contains user defined and hardware generated conditions that are used by MI_WAIT_FOR_EVENT commands. An MI_WAIT_FOR_EVENT instruction excludes the executing ring from arbitration if the selected event evaluates to a “1”, while instruction is discarded if the condition evaluates to a “0”. Once excluded a ring is enabled into arbitration when the selected condition evaluates to a “0”.

Bit	Description
31:16	**Mask Bits** Format: Mask[1] This bit serves as a write enable for bit 1. If this register is written with this bit clear the corresponding bit in the field 1 will not be modified. Reading these bits always returns 0s.
15:5	**Reserved** Project: All Format: MBZ
4:0	**User Defined Condition Codes** The software may signal a Stream Semaphore by setting the Mask bit and Signal Bit together to match the bit field specified in a WAIT_FOR_EVENT (Semaphore).

2.1.4.5 BRSYNC – Blitter/Render Semaphore Sync Register

BRSYNC – Blitter/Render Semaphore Sync Register	
Register Type:	MMIO_BCS
Address Offset:	22040h
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Trusted Type:	1
This register is written by CS, read by BCS.	
Bit	**Description**
31:0	**Semaphore Data** Semaphore data for synchronization between blitter engine and render engine..

2.1.4.6 BVSYNC – Blitter/Video Semaphore Sync Register

BVSYNC – Blitter/Video Semaphore Sync Register	
Register Type:	MMIO_BCS
Address Offset:	22044h
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Trusted Type:	1
This register is written by VCS, read by BCS.	
Bit	**Description**
31:0	**Semaphore Data** Semaphore data for synchronization between blitter engine and video codec engine.

2.1.4.7 GAB_MODE — Mode Register for GAB

Address Offset: 220A0h–220A3h

Default Value: 0000 0000h

Access: Read/Write

Size: 32 bits

The GAB_MODE register contains information that controls configurations in the GAB.

Bit	Description
31:16	**Masks:** A "1" in a bit in this field allows the modification of the corresponding bit in Bits 15:0
15:0	**Reserved** Read/Write

2.1.4.8 Logical Ring Context Format

The format of the logical ring context is documented in the "GPU Overview" volume, "Memory Data Formats" chapter.

2.1.4.9 Context Status

A context switch interrupt will be sent anytime a context switch occurs. This is documented in the "GPU Overview" volume, "Memory Data Formats" chapter. A status DW for the context that was just switched away from will be written to the Context Status Buffer in the Global Hardware Status Page. The status contains the context ID and the reason for the context switch. Note that since there will have been no running contexts when the very first (after reset) context is submitted, the Context ID in the first Context Status DWord will be UNDEFINED.

Bit	Description
31:12	**Context ID.** Contains the context ID copied from the submitted context.
11:8	Reserved: MBZ
7	Reserved: MBZ
6	Reserved: MBZ
5	Reserved: MBZ
4	**Ring Buffer Becoming Empty** Caused context to Switch.
3	Reserved: MBZ
2	Reserved: MBZ
1	**Waiting on a Semaphore** Caused Context to Switch.
0	Reserved: MBZ

2.1.4.10 BCS_RCCID—Ring Buffer Current Context ID Register

Address Offset: 22190h–22197h

Default Value: 00 00 00 00h

Access: Read/Write

Size: 32 bits

This register contains the current "ring context ID" associated with the ring buffer.

Programming Notes:

- The current context registers must not be written directly (via MMIO). The RCCID register should only be updated indirectly from RNCID.

Bit	Description
63:0	See Context Descriptor for BCS

2.1.4.11 VCS_RNCID—Ring Buff er Next Context ID Register

Address Offset: 22198h–2219fh

Default Value: 00 00 00 00h

Access: Read/Write

Size: 64 bits

This register contains the *next* "ring context ID" associated with the ring buffer.

Programming Notes:

- The current context (RCCID) register can be updated indirectly from this register on a context switch event. Note that the only time a context switch can occur is when MI_ARB_CHECK enables preemption or the current context runs dry (head pointer becomes equal to tail pointer).

Bit	Description
63:0	See Context Descriptor for BCS

2.1.4.12 Context Status

A context switch interrupt will be sent anytime a context switch occurs. This is documented in the "GPU Overview" volume, "Memory Data Formats" chapter. A status DW for the context that was just switched away from will be written to the Context Status Buffer in the Global Hardware Status Page. The status contains the context ID and the reason for the context switch. Note that since there will have been no running contexts when the very first (after reset) context is submitted, the Context ID in the first Context Status DWord will be UNDEFINED.

Bit	Description
31:12	**Context ID.** Contains the context ID copied from the submitted context.
11:8	Reserved: MBZ
7	Reserved: MBZ
6	Reserved: MBZ
5	Reserved: MBZ
4	**Ring Buffer Becoming Empty** Caused context to Switch.
3	Reserved: MBZ
2	Reserved: MBZ
1	**Waiting on a Semaphore** Caused Context to Switch.
0	Reserved: MBZ

2.1.5 BCS_RINGBUF—Ring Buffer Registers

RINGBUF—Ring Buffer Registers	
Register Type:	MMIO_BCS
Address Offset:	22030h
Project:	All
Default Value:	00000000h; 00000000h; 00000000h; 00000000h
Access:	R/W
Size (in bits):	4x32
Trusted Type:	1

These registers are used to define and operate the "ring buffer" mechanism which can be used to pass instructions to the command interface. The buffer itself is located in a physical memory region. The ring buffer is defined by a 4 Dword register set that includes starting address, length, head offset, tail offset, and control information. Refer to the *Programming Interface* chapter for a detailed description of the parameters specified in this ring buffer register set, restrictions on the placement of ring buffer memory, arbitration rules, and in how the ring buffer can be used to pass instructions.

Ring Buffer Head and Tail Offsets must be properly programmed before it is enabled. A Ring Buffer can be enabled when empty.

RINGBUF—Ring Buffer Registers

DWord	Bit	Description
0	31:21	Reserved Project: All Format: MBZ
	20:3	Tail Offset Project: All Format: U18 This field is written by software to specify where the valid instructions placed in the ring buffer end. The value written points to the QWord *past* the last valid QWord of instructions. In other words, it can be defined as the *next* QWord that software will write instructions into. Software must write subsequent instructions to QWords following the Tail Offset, possibly wrapping around to the top of the buffer (i.e., software can't skip around within the buffer). Note that all DWords prior to the location indicated by the Tail Offset must contain valid instruction data – which may require instruction padding by software. See Head Offset for more information. QWord Offset [DevSNB+] To work around a known hardware bug, the first command parsed when the tail is moved must always be an MI_BATCH_BUFFER_START command. Once inside, the batch buffer can be (but not required to be) immediately closed via MI_BATCH_BUFFER_END. [DevSNB+] Commands MI_NOOP * 32 MI_BATCH_BUFFER_START (only MI_BATCH_BUFFER_END inside batch) Blitter work MI_LOAD_REGISTER_IMM - addr: 0x2209c - data: 0x00020002 MI_FLUSH_DW w/ no post-sync op MI_LOAD_REGISTER_IMM - addr: 0x2209c - data: 0x00020000 MI_NOOP
	2:0	Reserved Project: All Format: MBZ

RINGBUF—Ring Buffer Registers		
1	31:21	Wrap Count Project: All Format: U11 This field is incremented by 1 whenever the Head Offset wraps from the end of the buffer back to the start (i.e., whenever it wraps back to 0). Appending this field to the Head Offset field effectively creates a virtual 4GB Head "Pointer" which can be used as a tag associated with instructions placed in a ring buffer. The Wrap Count itself will wrap to 0 upon overflow.
	20:2	Head Offset Project: All Format: U19 This field indicates the offset of the *next* instruction DWord to be parsed. Software will initialize this field to select the first DWord to be parsed once the RB is enabled. (Writing the Head Offset while the RB is enabled is UNDEFINED). Subsequently, the device will increment this offset as it executes instructions – until it reaches the QWord specified by the Tail Offset. At this point the ring buffer is considered "empty". Programming Notes: • A RB can be enabled empty or containing some number of valid instructions.
	1:0	Reserved Project: All Format: MBZ
2	31:12	Starting Address Project: All Format: Graphics Address[31:12] This field specifies Bits 31:12 of the 4KB-aligned starting Graphics Address of the ring buffer. Address bits 31 down to 29 must be zero. All ring buffer pages must map to Main Memory (uncached) pages. Ring Buffer addresses are always translated through the global GTT. Per-process address space can only be used via a batch buffer.
	11:0	Reserved Project: All Format: MBZ
3	31:21	Reserved Project: All Format: MBZ
	20:12	Buffer Length Project: All Format: U9 This field is written by SW to specify the length of the ring buffer in 4 KB Pages. Range = [0 = 1 page = 4 KB, 1FFh = 512 pages = 2 MB]
	11	RB Wait Project: All Format: Boolean Indicates that this ring has executed a WAIT_FOR_EVENT instruction and is currently waiting. Software can write a "1" to clear this bit, write of "0" has no effect. When the RB is waiting for an event and this bit is cleared, the wait will be terminated and the RB will be returned to arbitration.
	10	Semaphore Wait Project: DevSNB+ Format: Boolean Indicates that this ring has executed a MI_SEMAPHORE_MBOX instruction with register compare and is currently waiting. Software can write a "1" to clear this bit, write of "0" has no effect. When the RB is waiting for the compare to meet and this bit is cleared, the wait will be terminated and the RB will be returned to arbitration.
	9:3	Reserved Project: All Format: MBZ

RINGBUF—Ring Buffer Registers		
	2:1	Automatic Report Head Pointer Project: All Format: U2 This field is written by software to control the automatic “reporting” (write) of this ring buffer’s “Head Pointer” register (register DWord 1) to the corresponding location within the Hardware Status Page. Automatic reporting can either be disabled or enabled at 4KB, 64KB or 128KB boundaries within the ring buffer. Format = 0: MI_AUTOREPORT_OFF – Automatic reporting disabled 1: MI_AUTOREPORT_64KB – Report every 16 pages (64KB) 2: MI_AUTOREPORT_4KB – Report every page (4KB) 3: MI_AUTOREPORT_128KB – Report every 32 pages (128KB) When the Per-Process Virtual Address Space bit is set and automatic head reporting is desired, this field must be set to option 2 since the ring buffer will be only 16KB in size. The head pointer will be reported to the head pointer location in the PP HW Status Page when it passes each 4KB page boundary.
	0	Ring Buffer Enable Project: All Format: Enable This field is used to enable or disable this ring buffer. It can be enabled or disabled regardless of whether there are valid instructions pending.

2.1.5.1 BCS_UHPTR — Pending Head Pointer Register

BCS_UHPTR — Pending Head Pointer Register	
Register Type:	MMIO
Address Offset:	**22134h–22137h**
Project:	All
Security:	None
Default Value:	**0000 0000h**
Access:	R/W
Size (in bits):	32
Trusted Type:	1

Desc

Bit	Description
31:3	Head Pointer Address Project: All Security: None Default Value: 0h DefaultVaueDesc Mask: MMIO(0x2000)#16 Format: MI_Graphics_Offset FormatDesc Address: GraphicsAddress[31:0] Surface Type: U32 Range 0..2^32-1 This register represents the GFX address offset where execution should continue in the ring buffer following execution of an MI_ARB_CHECK command.
2:1	Reserved Project: All Format: MBZ
0	Head Pointer Valid Project: All Format: U32 This bit is set by the software to request a pre-emption. It is reset by hardware when an MI_ARB_CHECK command is parsed by the command streamer. The hardware uses the head pointer programmed in this register at the time the reset is generated. Value Name Description Project 1 Enable Indicates that there is an updated head pointer programmed in this register 0 Disable No valid updated head pointer register, resume execution at the current location in the ring buffer

2.1.6 Interrupt Control Registers

The Interrupt Control Registers described below all share the same bit definition. The bit definition is as follows:

Table 2-1. Bit Definition for Interrupt Control Registers

Bit	Description
31:30	**Reserved. MBZ:** These bits may be assigned to interrupts on future products/steppings.
29	**Page Fault:** This bit is set whenever there is a pending PPGTT (page or directory) fault.
28:27	Reserved. MBZ
26	**MI_FLUSH_DW Notify Interrupt:** The Pipe Control packet (Fences) specified in *3D pipeline* document may optionally generate an Interrupt. The Store QW associated with a fence is completed ahead of the interrupt.
25	**Blitter Command Parser Master Error:** When this status bit is set, it indicates that the hardware has detected an error. It is set by the device upon an error condition and cleared by a CPU write of a one to the appropriate bit contained in the Error ID register followed by a write of a one to this bit in the IIR. Further information on the source of the error comes from the "Error Status Register" which along with the "Error Mask Register" determine which error conditions will cause the error status bit to be set and the interrupt to occur. **Page Table Error:** Indicates a page table error. **Instruction Parser Error**: The Blitter Instruction Parser encounters an error while parsing an instruction.
24	**Sync Status:** This bit is set when the Instruction Parser completes a flush with the sync enable bit active in the INSTPM register. The event will happen after all the blitter engines are flushed. The HW Status DWord write resulting from this event will cause the CPU's view of graphics memory to be coherent as well (flush and invalidate the blitter cache). It is the driver's responsibility to clear this bit before the next sync flush with HWSP write enabled.
23	Reserved. MBZ
22	**Blitter Command Parser User Interrupt:** This status bit is set when an MI_USER_INTERRUPT instruction is executed on the Render Command Parser. Note that instruction execution is not halted and proceeds normally. A mechanism such as an MI_STORE_DATA instruction is required to associate a particular meaning to a user interrupt.
21:0	Reserved. MBZ

2.1.6.1 HWSTAM — Hardware Status Mask Register

Hardware Status Mask Register	
Register Type:	MMIO_BCS
Address Offset:	22098h
Project:	All
Default Value:	**FFFF FFFFh**
Access:	R/W, RO for Reserved Control bits
Size (in bits):	32
Trusted Type:	1

The HWSTAM register has the same format as the Interrupt Control Registers. The bits in this register are "mask" bits that prevent the corresponding bits in the Interrupt Status Register from generating a "Hardware Status Write" (PCI write cycle). Any unmasked interrupt bit (HWSTAM bit set to 0) will allow the Interrupt Status Register to be written to the ISR location (within the memory page specified by the Hardware Status Page Address Register) when that Interrupt Status Register bit changes state.

Programming Note:

- To write the interrupt to the HWSP, the corresponding IMR bit must also be clear (enabled).
- At most 1 bit can be unmasked at any given time

Bit	Description
31:0	**Hardware Status Mask Register** Project: All Default Value: FFFFFFFFh DefaultVaueDesc Format: Array of Masks Refer to Table 5-1 in Interrupt Control Register section for bit definitions

2.1.6.2 IMR—Interrupt Mask Register

IMR—Interrupt Mask Register	
Register Type:	MMIO_BCS
Address Offset:	220A8h
Project:	All
Default Value:	FFFF FFFFh
Access:	R/W
Size (in bits):	32

The IMR register is used by software to control which Interrupt Status Register bits are “masked” or “unmasked”. “Unmasked” bits will be reported in the IIR, possibly triggering a CPU interrupt, and will persist in the IIR until cleared by software. “Masked” bits will not be reported in the IIR and therefore cannot generate CPU interrupts.

Bit	Description
31:0	**Interrupt Mask Bits** Project: All Default Value: FFFF FFFFh Format: Array of interrupt mask bits — Refer to Table 5-1 in Interrupt Control Register section for bit definitions This field contains a bit mask which selects which interrupt bits (from the ISR) are reported in the IIR.

Value	Name	Description	Project
0h	Not Masked	Will be reported in the IIR	All
1h	Masked	Will not be reported in the IIR	All

2.1.6.3 Hardware-Detected Error Bit Definitions (for EIR, EMR, ESR)

This section defines the Hardware-Detected Error bit definitions and ordering that is common to the EIR, EMR and ESR registers. The EMR selects which error conditions (bits) in the ESR are reported in the EIR. Any bit set in the EIR will cause the Master Error bit in the ISR to be set. EIR bits will remain set until the appropriate bit(s) in the EIR is cleared by writing the appropriate EIR bits with '1' (except for the unrecoverable bits described below).

The following table describes the Hardware-Detected Error bits:

Table 2-2. Hardware-Detected Error Bits

Bit	Description
15:1	Reserved: MBZ
0	**Instruction Error:** This bit is set when the Renderer Instruction Parser detects an error while parsing an instruction. Instruction errors include: 1) Client ID value (Bits 31:29 of the Header) is not supported (only MI, 2D and 3D are supported). 2) Defeatured MI Instruction Opcodes: 1: Instruction Error detected Programming Note: This error indication can not be cleared except by reset (i.e., it is a fatal error).

2.1.6.3.1 EIR — Error Identity Register

EIR — Error Identity Register

Register Type: MMIO_BCS
Address Offset: 220B0h
Project: All
Default Value: 0000 0000h
Access: R/WC
Size (in bits): 32

The EIR register contains the persistent values of Hardware-Detected Error Condition bits. Any bit set in this register will cause the Master Error bit in the ISR to be set. The EIR register is also used by software to clear detected errors (by writing a '1' to the appropriate bit(s) except for the unrecoverable bits described).).

Bit	Description
31:16	**Reserved** Project: All Format: MBZ
15:0	**Error Identity Bits** Project: All Default Value: 0h Format: Array of Error condition bits See Table 1 5. Hardware-Detected Error Bits This register contains the persistent values of ESR error status bits that are unmasked via the EMR register. (See **Error! Reference source not found.**). The logical OR of all (defined) bits in this register is reported in the Master Error bit of the Interrupt Status Register. In order to clear an error condition, software must first clear the error by writing a '1' to the appropriate bit(s) in this field. If required, software should then proceed to clear the Master Error bit of the IIR. **Value** / **Name** / **Description** / **Project** 1h / Error occurred / Error occurred / All **Programming Notes** — **Project** Writing a '1' to a set bit will cause that error condition to be cleared. However, the Instruction Error bit (Bit 0) cannot be cleared except by reset (i.e., it is a fatal error). — All

2.1.6.3.2 EMR—Error Mask Register

EMR—Error Mask Register

Register Type: MMIO_BCS
Address Offset: 220B4h
Project: All
Default Value: FFFF FFFFh
Access: R/W
Size (in bits): 32

The EMR register is used by software to control which Error Status Register bits are “masked” or “unmasked”. “Unmasked” bits will be reported in the EIR, thus setting the Master Error ISR bit and possibly triggering a CPU interrupt, and will persist in the EIR until cleared by software. “Masked” bits will not be reported in the EIR and therefore cannot generate Master Error conditions or CPU interrupts.

Bit	Description
31:16	**Reserved** Project: All Format: MBZ
15:0	**Error Mask Bits** Project: All Default Value: FFFF FFFFh Format: Array of error condition mask bits See Table 1 5. Hardware-Detected Error Bits This register contains a bit mask that selects which error condition bits (from the ESR) are reported in the EIR.

Value	Name	Description	Project
0h	Not Masked	Will be reported in the EIR	All
1h	Masked	Will not be reported in the EIR	All

2.1.6.3.3 ESR—Error Status Register

ESR—Error Status Register	
Register Type:	MMIO_BCS
Address Offset:	220B8h
Project:	All
Default Value:	0000 0000h
Access:	RO
Size (in bits):	32

The ESR register contains the current values of all Hardware-Detected Error condition bits (these are all by definition "persistent"). The EMR register selects which of these error conditions are reported in the persistent EIR (i.e., set bits must be cleared by software) and thereby causing a Master Error interrupt condition to be reported in the ISR.

Bit	Description
31:16	Reserved Project: All Format: MBZ
15:0	Error Status Bits Project: All Default Value: 0h Format: Array of error condition bits See Table 1 5. Hardware-Detected Error Bits This register contains the non-persistent values of all hardware-detected error condition bits.

Value	Name	Description	Project
1h	Error Condition Detected	Error Condition detected	All

2.1.7 Logical Context Support

2.1.7.1 BCS_BB_ADDR—Batch Buffer Head Pointer Register

Address Offset: 022140h–022147h

Default Value: 0000 0000 0000 0000h

Access: Read-Only

Size: 64 bits

This register contains the current QWord Graphics Memory Address of the last-initiated batch buffer.

Bit	Description
63:32	Reserved: MBZ
31:3	**Batch Buffer Head Pointer:** This field specifies the QWord-aligned Graphics Memory Address where the last initiated Batch Buffer is currently fetching commands. If no batch buffer is currently active, the Valid bit will be 0 and this field will be meaningless. .
2:1	Reserved: MBZ
0	Valid: 1 = Batch buffer Valid 0 = Batch buffer Invalid

2.1.8 Software Control Bit Definitions

Registers in the range 22XX are not protected from the load register immediate instruction if the command is executed in the non-secure batch buffer.

2.1.8.1 TIMESTAMP — Reported Timestamp Count

TIMESTAMP — Reported Timestamp Count	
Register Type:	MMIO_BCS
Address Offset:	22358h
Project:	All
Default Value:	0000 0000 0000 0000h
Access:	RO. This register is *not* set by the context restore.
Size (in bits):	64

This register provides an elapsed real-time value that can be used as a timestamp.

This register is *not* reset by a graphics reset. It will maintain its value unless a full chipset reset is performed.

Bit	Description
63:36	Reserved Project: All Format: MBZ
35:0	Timestamp Value PrProject: All Format: U 32 This register toggles every 640 ns of time.

2.2 Memory Interface Commands for Blitter Engine

2.2.1 Introduction

This chapter describes the formats of the “Memory Interface” commands, including brief descriptions of their use. The functions performed by these commands are discussed fully in the *Memory Interface Functions* Device Programming Environment chapter.

This chapter describes MI Commands for the blitter graphics processing engine. The term “for Blitter Engine” in the title has been added to differentiate this chapter from a similar one describing the MI commands for the Media Decode Engine and the Rendering Engine.

The commands detailed in this chapter are used across products within the Gen4 family. However, slight changes may be present in some commands (i.e., for features added or removed), or some commands may be removed entirely. Refer to the *Preface* chapter for product specific summary.

2.2.2 MI_ARB_CHECK

The instruction format is:

MI_ARB_CHECK			
Project:	All	**Length Bias:**	1
Engine:	**Blitter**		

The MI_ARB_CHECK instruction is used to check the ring buffer next context ID register (RNCID) or the UHPTR register, depending on whether PPGTT/Runlists are enabled. This instruction can be used to pre-empt the current execution of the ring buffer. Note that the valid bit in the RNCID register or the UHPTR register needs to be set for the command streamer to be pre-empted.

Programming Note:

This instruction can be placed only in a ring buffer, never in a batch buffer.

DWord	Bit	Description
0	31:29	Command Type Default Value: 0h MI_INSTRUCTION Format: OpCode
	28:23	MI Command Opcode Default Value: 05h MI_ARB_CHECK Format: OpCode
	22:0	Reserved Project: All Format: MBZ

2.2.3 MI_BATCH_BUFFER_END

MI_BATCH_BUFFER_END			
Project:	DevSNB+	**Length Bias:**	1
Engine:	**Blitter**		

The MI_BATCH_BUFFER_END command is used to terminate the execution of commands stored in a *batch buffer* initiated using a MI_BATCH_BUFFER_START command.

DWord	Bit	Description
0	31:29	Command Type Default Value: 0h MI_COMMAND Format: OpCode
	28:23	MI Command Opcode Default Value: 0Ah MI_ BATCH_BUFFER_END Format: OpCode
	22:0	Reserved Project: All Format: MBZ

2.2.4 MI_BATCH_BUFFER_START

MI_BATCH_BUFFER_START			
Project:	DevSNB+	**Length Bias:**	2
Engine:	**Blitter**		

The MI_BATCH_BUFFER_START command is used to initiate the execution of commands stored in a *batch buffer.* For restrictions on the location of batch buffers, see Batch Buffers in the Device Programming Interface chapter of *MI Functions*.

The batch buffer can be specified as secure or non-secure, determining the operations considered valid when initiated from within the buffer and any attached (chained) batch buffers. See Batch Buffer Protection in the Device Programming Interface chapter of *MI Functions*.

Programming Notes:

- Batch buffers referenced with physical addresses must not extend beyond the end of the starting physical page (can't span physical pages). However, a batch buffer initiated using a physical address can chain to another buffer in another physical page.
- A batch buffer initiated with this command must end either with a MI_BATCH_BUFFER_END command or by chaining to another batch buffer with an MI_BATCH_BUFFER_START command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	MI Command Opcode Default Value: 31h MI_BATCH_BUFFER_START Format: OpCode
	22	Reserved
	22	Reserved
	21:9	Reserved

	8	Buffer Security Indicator Project: DevSNB+ Format: MI_BufferSecurityType When this command is executed from within a batch buffer (i.e., is a “chained” batch buffer command), this field is IGNORED and the next buffer in the chain inherits the initial buffer’s security characteristics. [DevSNB] If this bit is set, this batch buffer is non-secure and cannot execute privileged commands nor access privileged (GGTT) memory. It will be accessed via the PPGTT. If clear, this batch buffer is secure and will be accessed via the GGTT. Note that MI_STORE_DATA_IMM to non-privileged memory (via the PPGTT) *is* allowed in a non-secure batch buffer.

Value	Name	Description	Project
0h	GGTT	This batch buffer is secure and will be accessed via the GGTT.	All
1h	PPGTT	This batch buffer is always treated as non-secure and cannot execute privileged commands nor access privileged (GGTT) memory. It will always be accessed via the PPGTT.	All

Programming Notes

This field must be ‘0’ unless the **Per-Process GTT Enable** is ‘1’

7:0	DWord Length		
	Default Value:	0h	Excludes DWord (0,1)
	Format:	=n	Total - Bias

MI_BATCH_BUFFER_START

	31:0	**BitFieldName** Project: All Security: None Access: None Exists If: Always Default Value: 0h DefaultVaueDesc Mask: MMIO(0x2000)#16 Format: U32 FormatDesc Address: GraphicsAddress[31:0] Surface Type: U32 Range 0..2^32-1 BitFieldDesc

Value	Name	Description	Project
0h	Disable	Desc	All
1h	Enable	Desc	All

1	31:2	**Batch Buffer Start Address** Project: All Address: GraphicsAddress[31:2] Surface Type: BatchBuffer This field specifies Bits 31:2 of the starting address of the batch buffer.
	1:0	Reserved Project: All Format: MBZ

2.2.5 MI_FLUSH_DW

MI_FLUSH_DW			
Project:	DevSNB+	**Length Bias:**	2
Engine:	**Blitter**		

The MI_FLUSH_DW command is used to perform an internal "flush" operation. The parser pauses on an internal flush until all drawing engines have completed any pending operations. In addition, this command can also be used to:

Flush any dirty data to memory.

Invalidate the TLB cache inside the hardware

Usage note**: After this command is completed with a Store DWord enabled, CPU access to graphics memory will be coherent (assuming the Render Cache flush is not inhibited).**

[DevSNB+] An MI_NOOP with NOP_ID bit set must be programmed after the last MI_FLUSH_DW before head = tail

DWord	Bit	Description
0	31:29	Command Type Default Value: 0h MI_COMMAND Format: OpCode
	28:23	MI Command Opcode Default Value: 26h MI_FLUSH_DW Format: OpCode
	22	Reserved Project: All Format: U1
	21	Store Data Index Project: DevSNB+ Format: U1 This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is actually an index into the hardware status page. If this bit is set, this command will index into the per-process hardware status page if executed from within a non-secure batch buffer and if the Per-Process Virtual Address Space bit is set. Else the Global HW status page is used.
	20:19	Reserved Project: All Format: MBZ
	18	TLB Invalidate Project: DevSNB+ Format: U1 If ENABLED, all TLBs will be invalidated once the flush operation is complete. Note that if the flush TLB invalidation mode is clear, a TLB invalidate will occur irrespective of this bit setting. This bit is only valid when the Post-Sync Operation field is a value of 1h or 3h.
	17	Synchronize GFDT surface Project: DevSNB+ Format: U1 If enabled, at the end of the current flush the last level cache is cleared of all the cachelines which have been marked with the special GFDT flags. Store DW must be enabled
	16	Reserved Project: All Format: MBZ

MI_FLUSH_DW

	15:14	Post-Sync Operation Project: DevSNB+ BitFieldDesc **Value** **Name** **Description** **Project** 0h No write occurs as a result of this instruction. This can be used to implement a “trap” operation, etc. DevSNB+ 1h Write the QWord containing Immediate Data Low, High DWs to the Destination Address DevSNB+ 2h Reserved DevSNB+ 3h Write the 32-bit TIMESTAMP register to the Destination Address with granularity of 640ns. Upper 32-bits are tied to ‘0’ DevSNB+ Programming Notes If executed in non-secure batch buffer, the address given will be in a PPGTT address space. If in a secure ring or batch, address given will be in GGTT space
	13:9	Reserved Project: All Format: MBZ
	8	Notify Enable Project: DevSNB+ Format: U1 If ENABLED, a Sync Completion Interrupt will be generated (if enabled by the MI Interrupt Control registers) once the sync operation is complete. See Interrupt Control Registers in *Memory Interface Registers* for details.
	7:6	Reserved Project: All Format: MBZ
	5:0	DWord Length Default Value: 2h Excludes DWord (0,1) = 1 for DWord, 2 for QWord Format: =n Total Length - 2 Project: All
1	31:3	Address Project: DevSNB+ Address: GraphicsAddress[31:2] Surface Type: U32 This field specifies Bits 31:3 of the Address where the DWord or QWord will be stored. Note that the address can only be QWord aligned, irrespective of data size.

MI_FLUSH_DW		
	2	Destination Address Type Project: All Defines address space of Destination Address Value / Name / Description / Project 0h / PPGTT / Use PPGTT address space for DW write / All 1h / GGTT / Use GGTT address space for DW write / All **Programming Notes** Ignored if "No write" is the selected in Operation.
	1:0	Reserved Project: All Format: MBZ
2..3	31:0	Immediate Data Format: U32 Address: GraphicsAddress[31:0] Range 0..2^32-1 This field specifies the DWord value to be written to the targeted location. DW2 is the lower DW if QW is desired. Only valid when 15:14 in header is set to 1h

2.2.6 MI_LOAD_REGISTER_IMM

MI_LOAD_REGISTER_IMM			
Project:	DevSNB+	**Length Bias:**	2
Engine:	**Blitter**		

The MI_LOAD_REGISTER_IMM command requests a write of up to a DWord constant supplied in the command to the specified Register Offset (i.e., offset into Memory-Mapped Register Range). The register is loaded before the next command is executed.

Programming Notes:

[DevSNB]The behavior of this command is controlled by Dword 3, Bit 8 (**Disable Register Access**) of the RINGBUF register. If this command is disallowed then the command stream converts it to a NOOP.

If this command is executed from a BB then the behavior of this command is controlled by Dword 0, Bit 8 (**Security Indicator**) of the BATCH_BUFFER_START Command. If the batch buffer is insecure then the command stream converts this command to a NOOP. Note that the corresponding ring buffer must allow a register update for this command to execute.

The following addresses should NOT be used for LRIs

1. 0x8800 - 0x88FF
2. >= 0x40000

DWord	Bit	Description
0	31:29	Command Type Default Value: 0h MI_COMMAND Format: OpCode
	28:23	MI Command Opcode Default Value: 22h MI_ Format: OpCode
	22:12	Reserved Project: All Format: MBZ
	11:8	Byte Write Disables Format: Enable[4] Bit 8 corresponds to Data DWord [7:0] Range Must specify a valid register write operation If [11:8] is ‘1111’, then the register write will not occur. If [11:8] is '0000', then the register DW will be updated. Any other value, the behavior will be specifically specified by the register or the behavior is undefined.
	7:0	DWord Length Default Value: 1h Excludes DWord (0,1) Format: =n Total Length - 2

MI_LOAD_REGISTER_IMM		
1	31:2	Register Offset Format: U30 Address: MmioAddress[31:2] This field specifies bits [31:2] of the offset into the Memory Mapped Register Range (i.e., this field specifies a DWord offset).
	1:0	Reserved Project: All Format: MBZ
2	31:0	Data DWord Mask: Bytes Write Disables Format: U32 This field specifies the DWord value to be written to the targeted location.

2.2.7 MI_NOOP

MI_NOOP			
Project:	DevSNB+	**Length Bias:**	1
Engine:	**Blitter**		

The MI_NOOP command basically performs a “no operation” in the command stream and is typically used to pad the command stream (e.g., in order to pad out a batch buffer to a QWord boundary). However, there is one minor (optional) function this command can perform – a 22-bit value can be loaded into the MI NOPID register. This provides a general-purpose command stream tagging ("breadcrumb") mechanism (e.g., to provide sequencing information for a subsequent breakpoint interrupt).

DWord	Bit	Description
0	31:29	Command Type Default Value: 0h MI_COMMAND Format: OpCode
	28:23	MI Command Opcode Default Value: 0h MI_NOOP Format: OpCode
	22	Identification Number Register Write Enable Project: All Format: Enable This field enables the value in the Identification Number field to be written into the MI NOPID register. If disabled, that register is unmodified – making this command an effective “no operation” function. Value / Name / Description / Project 0h / Disable / Do not write the NOP_ID register. / All 1h / Enable / Write the NOP_ID register. / All
	31:0	Identification Number Project: All Format: U22 This field contains a 22-bit number which can be written to the MI NOPID register.

2.2.8 MI_REPORT_HEAD

MI_REPORT_HEAD			
Project:	DevSNB+	**Length Bias:**	1
Engine:	**Blitter**		

The MI_REPORT_HEAD command causes the Head Pointer value of the active ring buffer to be written to a cacheable (snooped) system memory location.

Programming Notes:

- This command must not be executed from a Batch Buffer (Refer to the description of the HSW_PGA register).

DWord	Bit	Description
0	31:29	Command Type Default Value: 0h MI_COMMAND Format: OpCode
	28:23	MI Command Opcode Default Value: 07h MI_REPORT_HEAD Format: OpCode
	22:0	Reserved Project: All Format: MBZ

2.2.9 MI_SEMAPHORE_MBOX

MI_SEMAPHORE_MBOX			
Project:	DevSNB+	**Length Bias:**	2
Engine:	**Blitter**		

This command is provided as alternative to MI_SEMAPHORE to provide mailbox-type semaphores where there is no update of the semaphore by the checking process (the consumer). Single-bit compare-and-update semantics are also provided. In either case, atomic access of semaphores need not be guaranteed by hardware as with the previous command. This command should eventually supersede the previous command.

Synchronization between contexts (especially between contexts running on 2 different engines) is provided by the MI_SEMAPHORE_MBOX command. Note that contexts attempting to synchronize in this fashion must be able to access a common memory location. This means the contexts must share the same virtual address space (have the same page directory), must have a common physical page mapped into both of their respective address spaces or the semaphore commands must be executing from a secure batch buffer or directly from a ring with the **Use Global GTT** bit set such that they are "privileged" and will use the (always shared) global GTT.

MI_SEMAPHORE with the **Update Semaphore** bit set (and the **Compare Semaphore** bit clear) implements the *Signal* command, while the *Wait* command is indicated by **Compare Semaphore** being set. Note that *Wait* can cause a context switch. *Signal* increments unconditionally.

MI_SEMAPHORE_MBOX		
DWord	**Bit**	**Description**
0	31:29	Command Type Default Value: 0h MI_COMMAND Format: OpCode
	28:23	MI Command Opcode Default Value: 16h MI_SEMAPHORE_MBOX Format: OpCode
	22	Use Global GTT Project: All Format: U32 If set, this command will use the global GTT to translate the Semaphore Address and this command must be executing from a privileged (secure) batch buffer. If clear, the PPGTT will be used to translate the Semaphore Address. This bit will be ignored (and treated as if clear) if this command is executed from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer or directly from a ring buffer.
	21	Update Semaphore Project: All Format: U32 If set, the value from the Semaphore Data Dword is written to memory. If Compare Semaphore is also set, the semaphore is not updated if the semaphore comparison fails. If clear, the data at Semaphore Address is not changed.
	20	Compare Semaphore Project: All Format: U32 If set, the value from the Semaphore Data Dword is compared to the value from the Semaphore Address in memory. If the value at Semaphore Address is greater than the Semaphore Data Dword, execution is continued from the current command buffer. If clear, no comparison takes place. Update Semaphore *must* be set in this case.
	19	Reserved Project: All Format: MBZ
	18	Compare Register Project: DevSNB+ Format: Compare Type If set, data in MMIO register will be used for compare. If clear, data in memory will be used for compare.
	17:16	Register Select Project: DevSNB+ Format: Register Select If compare register is set in bit[18], this filed indicate which register will be used. 0: CS register (BRSYNC) 1: Reserved 2: VCS regiser (BVSYNC) 3: Reserved
	15:8	Reserved Project: All Format: MBZ
	7:0	DWord Length Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2

MI_SEMAPHORE_MBOX		
1	31:0	Semaphore Data Dword Project: All Format: U32 Data dword to compare/update memory. The Data dword is supplied by software to control execution of the command buffer. If the compare is enabled and the data at Semaphore Address is greater than this dword, the execution of the command buffer continues.
2	31:2	PointerBitFieldName/MMIO Register Address Project: All Address: GraphicsVirtualAddress[31:2] Surface Type: Semaphore if Compare Register bit[18] is cleared, this field if the Graphics Memory Address of the 32 bit value for the semaphore. If Compare Register bit[18] is set, this field is the MMIO address of the register for the semaphore.
	1:0	Reserved Project: All Format: MBZ

2.2.10 MI_STORE_REGISTER_MEM

MI_STORE_REGISTER_MEM			
Project:	DevSNB+	**Length Bias:**	2
Engine:	Blitter		

The MI_STORE_REGISTER_MEM command requests a register read from a specified memory mapped register location in the device and store of that DWord to memory. The register address is specified along with the command to perform the read.

Programming Notes:

The command temporarily halts command execution.

The memory address for the write is snooped on the host bus.

This command will cause undefined data to be written to memory if given register addresses for the PGTBL_CTL_0 or FENCE registers

[DevSNB+]: To avoid deadlock scenarios, this command cannot be executed if there are additional posted writes (i.e. LRI, semaphore update) being sent to the same command streamer.

The following addresses should NOT be used for SRMs

```
1. 0x8800 - 0x88FF

2. >= 0x40000
```

The only exception is an SRM cycle to 0x40000-0xBFFFF when used as part of the LRI read-after-write requirement.

MI_STORE_REGISTER_MEM

DWord	Bit	Description
0	31:29	Command Type Default Value: 0h MI_COMMAND Format: OpCode
	28:23	MI Command Opcode Default Value: 24h MI_STORE_REGISTER_MEM Format: OpCode
	22	Use Global GTT Project: All This bit *must* be '1' if the Per Process GTT Enable bit is clear Value / Name / Description / Project 0h / Per Process Graphics Address / / All 1h / Global Graphics Address / This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer. / All Programming Notes: [DevSNB] This will be ignored and treated as if clear when executing from a PPGTT (i.e. runlist mode "non-secure") batch buffer
	21:8	Reserved Project: All Format: MBZ
	7:0	DWord Length Default Value: 1h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:23	Reserved Project: DevSNB+ Format: MBZ
	22:2	Register Address Project: All Address: MMIO Address[22:2] Surface Type: MMIO Register This field specifies Bits 22:2 of the Register offset the DWord will be read from. As the register address must be DWord-aligned, Bits 1:0 of that address MBZ. Programming Notes Storing a VGA register is not permitted and will store an UNDEFINED value. The values of PGTBL_CTL0 or any of the FENCE registers cannot be stored to memory; UNDEFINED values will be written to memory if the addresses of these registers are specified.
	1:0	Reserved Project: All Format: MBZ

MI_STORE_REGISTER_MEM		
2	31:2	Memory Address Project: DevSNB+ Address: GraphicsAddress[31:2] Surface Type: MMIO Register Range: GraphicsVirtualAddress[31:2] for a DWord register This field specifies the address of the memory location where the register value specified in the DWord above will be written. The address specifies the DWord location of the data.
	1:0	Reserved Project: All Format: MBZ

2.2.11 MI_STORE_DATA_IMM

MI_STORE_DATA_IMM			
Project:	DevSNB+	**Length Bias:**	2
Engine:	**Blitter**		

The MI_STORE_DATA_IMM command requests a write of the QWord constant supplied in the packet to the specified Memory Address. As the write targets a System Memory Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes:

This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll un-cached memory or device registers). However, the cacheable nature of the transaction is determined by the setting of the “mapping type” in the GTT entry.

This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete “eventually”, there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations. All writes to memory generated using this command are expected to finish in order.

DWord	Bit	Description
0	31:29	Command Type Default Value: 0h MI_COMMAND Format: OpCode
	28:23	MI Command Opcode Default Value: 20h MI_STORE_DATA_IMM Format: OpCode

MI_STORE_DATA_IMM		
	22	Use Global GTT Project: All This bit *must* be '1' if the Per Process GTT Enable bit is clear. Value / Name / Description / Project 0h / Per Process Graphics Address / / All 1h / Global Graphics Address / This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer. / All Programming Notes: [DevSNB] This will be ignored and treated as if clear when executing from a PPGTT (i.e. runlist mode "non-secure") batch buffer
	21:8	Reserved Project: All Format: MBZ
	7:0	DWord Length Default Value: 2h Excludes DWord (0,1) = 2 for DWord, 3 for QWord Format: =n Total Length - 2
1	31:0	Reserved Project: All Format: MBZ
2	31:2	Address Project: All Address: GraphicsAddress[31:2] Surface Type: U32(2) This field specifies Bits 31:2 of the Address where the DWord will be stored. As the store address must be DWord-aligned, Bits 1:0 of that address MBZ. This address must be 8B aligned for a store "QW" command.
	1:0	Reserved Project: All Format: MBZ
3	31:0	Data DWord 0 Project: All Format: U32 This field specifies the DWord value to be written to the targeted location. For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).
4	31:0	Data DWord 1 Project: All Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

2.2.12 MI_STORE_DATA_INDEX

MI_STORE_DATA_INDEX

MI_STORE_DATA_INDEX

Project:	DevSNB+	**Length Bias:**	2
Engine:	**Blitter**		

The MI_STORE_DATA_INDEX command requests a write of the data constant supplied in the packet to the specified offset from the System Address defined by the Hardware Status Page Address Register. As the write targets a System Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes:

Use of this command with an invalid or uninitialized value in the Hardware Status Page Address Register is UNDEFINED.

This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll uncached memory or device registers).

This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete "eventually", there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

DWord	Bit	Description
0	31:29	Command Type Default Value: 0h MI_COMMAND Format: OpCode
	28:23	MI Command Opcode Default Value: 21h MI_STORE_DATA_INDEX Format: OpCode
	22	Reserved Project: All Format: MBZ
	21	Use Per-Process Hardware Status Page Project: All If this bit is set, this command will index into the per-process hardware status page at offset 28K from the LRCA. If clear, the Global Hardware Status Page will be indexed. Programming Notes: [DevSNB] This will be ignored and treated as if set when executing from a PPGTT (i.e. runlist mode "non-secure") batch buffer
	20:8	Reserved Project: All Format: MBZ
	7:0	DWord Length Default Value: 1h Excludes DWord (0,1) = 1 for DWord, 2 for QWord Format: =n Total Length - 2
1	31:12	Reserved Project: All Format: MBZ

MI_STORE_DATA_INDEX		
	11:2	Offset Project: All Format: U10 zero-based DWord offset into the HW status page. Address: HardwareStatusPageOffset[11:2] Surface Type: U32 Range [16, 1023] This field specifies the offset (into the hardware status page) to which the data will be written. Note that the first few DWords of this status page are reserved for special-purpose data storage – targeting these reserved locations via this command is UNDEFINED. This address must be 8B aligned for a store "QW" command.
	1:0	Reserved Project: All Format: MBZ
2	31:0	Data DWord 0 Project: All Format: U32 This field specifies the DWord value to be written to the targeted location. For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).
3	31:0	Data DWord 1 Project: All Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

2.2.13 MI_SUSPEND_FLUSH

MI_SUSPEND_FLUSH			
Project:	All	**Length Bias:**	1
Engine:	**Blitter**		
Blocks MMIO sync flush or any flushes related to VT-d while enabled.			
DWord	**Bit**	**Description**	
0	31:29	Command Type Default Value: 0h MI_COMMAND Format: OpCode	
	28:23	MI Command Opcode Default Value: 0Bh MI_SUSPEND_FLUSH Format: OpCode	
	22:1	Reserved Project: All Format: MBZ	

MI_SUSPEND_FLUSH		
	0	Suspend Flush Project: All Default Value: 0h DefaultVaueDesc Format: Enable FormatDesc This field suspends flush due to sync flush or implicit flush generated during VTD enable, disable and IOTLB invalidation. **Value** **Name** **Description** **Project** 0h Disable All 1h Enable All

2.2.14 MI_UPDATE_GTT

2.2.14.1 MI_UPDATE_GTT [DevSNB]

MI_UPDATE_GTT			
Project:	[DevSNB]	**Length Bias:**	2
Engine:	Blitter		

The MI_UPDATE_GTT command is used to update GTT page table entries in a coherent manner and at a predictable place in the command flow.

An MI_FLUSH should be placed before this command, since work associated with preceding commands that are still in the pipeline may be referencing GTT entries that will be changed by its execution. The flush will also invalidate TLBs and read caches that may become invalid as a result of the changed GTT entries. MI_FLUSH is not required if it can be guaranteed that the pipeline is free of any work that relies on changing GTT entries (such as MI_UPDATE_GTT contained in a paging DMA buffer that is doing only update/mapping activities and no rendering).

This is a privileged command. This command will be converted to a no-op and an error flagged if it is executed from within a non-secure batch buffer.

MI_UPDATE_GTT contents must be in address/data pair. This is different from the render CS definition. PPGTT updates cannot be done via MI_UPDATE_GTT, gfx driver will have to use storeDW for PPGTT inline updates.

[DevSNB] The address of the header cannot be 64-bit aligned.

Note that MI_UPDATE_GTT is mainly for the pages that are strictly used by processor graphics. If driver chooses to update the CPU used pages thru MI_UPDATE_GTT, it needs to write to MMIO address x101008 (any value) to ensure system agent TLBs are invalidated before the new pages can be used.

DWord	Bit	Description
0	31:29	Command Type Default Value: 0h MI_COMMAND Format: OpCode

MI_UPDATE_GTT

	28:23	MI Command Opcode Default Value: 23h MI_UPDATE_GTT Format: OpCode
	22	Use Global GTT Project: All Reserved: Must be 1h. Updating Per Process Graphics Address is not supported Value / Name / Description / Project 0h / Per Process Graphics Address / Illegal, not supported. / All 1h / Global Graphics Address / This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer. / All
	21: 6	Reserved Project: All Format: MBZ
	5:0	DWord Length Default Value: 0h Excludes DWord (0,1) Format: = n Total Length - 2, max 61
1..n+1	63:44	Entry Address Project: All Address: GraphicsAddress[31:12] This field simply holds the DW offset of the first table entry to be modified. Note that one or more of the upper bits may need to be 0, i.e., for a 2G aperture, bit 31 MBZ.
	43:32	Reserved Project: All Format: MBZ
	31:0	Entry Data Project: All Format: Table Entry This Dword becomes the new page table entry. See PPGTT/Global GTT Table Entries (PTEs) in Memory Interface Registers.

2.2.15 MI_USER_INTERRUPT

MI_USER_INTERRUPT			
Project:	DevSNB+	**Length Bias:**	1
Engine:	**Blitter**		

The MI_USER_INTERRUPT command is used to generate a User Interrupt condition. The parser will continue parsing after processing this command. See User Interrupt.

DWord	Bit	Description
0	31:29	Command Type Default Value: 0h MI_COMMAND Format: OpCode
	28:23	MI Command Opcode Default Value: 02h MI_USER_INTERRUPT Format: OpCode
	22:0	Reserved Project: All Format: MBZ

2.2.16 MI_WAIT_FOR_EVENT [DevSNB]

MI_WAIT_FOR_EVENT			
Project:	[DevSNB]	**Length Bias:**	1
Engine:	**Blitter**		

The MI_WAIT_FOR_EVENT command is used to pause command stream processing until a specific event occurs or while a specific condition exists. See Wait Events/Conditions, Device Programming Interface in *MI Functions*. Only one event/condition can be specified -- specifying multiple events is UNDEFINED.

The effect of the wait operation depends on the source of the command. If executed from a batch buffer, the parser will halt (and suspend command arbitration) until the event/condition occurs. If executed from a ring buffer, further processing of that ring will be suspended, although command arbitration (from other rings) will continue. Note that if a specified condition does not exist (the condition code is inactive) at the time the parser executes this command, the parser proceeds, treating this command as a no-operation.

If execution of this command from a primary ring buffer causes a wait to occur, the active ring buffer will *effectively* give up the remainder of its time slice (required in order to enable arbitration from other primary ring buffers).

DWord	Bit	Description
0	31:29	Command Type Default Value: 0h MI_COMMAND Format: OpCode
	28:23	MI Command Opcode Default Value: 03h MI_WAIT_FOR_EVENT Format: OpCode
	22:20	Reserved Project: All Format: MBZ

MI_WAIT_FOR_EVENT		
	19:16	Condition Code Wait Select Project: All This field enables a wait for the duration that the corresponding condition code is active. These enable select one of 15 condition codes in the EXCC register, that cause the parser to wait until that condition-code in the EXCC is cleared. **Value** / **Name** / **Description** / **Project** 0h / Not enabled / Condition Code Wait Not Enabled / All 1h-5h / Enable / Condition Code select enabled; selects one of 5 codes, 0 – 4 / All 6h-15h / Reserved / All Programming Notes Note that not all condition codes are implemented. The parser operation is UNDEFINED if an unimplemented condition code is selected by this field. The description of the EXCC register (*Memory Interface Registers*) lists the codes that are implemented.
	15:0	Reserved Project: All Format: MBZ

2.2.17 MSG_LOAD_SL – Load Scan Lines Message

MSG_LOAD_SL—Load Scan Lines Message	
Register Type:	Inbound Message
Address Offset:	Pipe A: 50020h Pipe B: 50028h Pipe C: 5002Ch
Project:	All
Access:	Sent from PG to North Display on Primary Channel
Size (in bits):	32

This message is used to initiate a display scan line compare.

When this message is received with the Init_Compare bit set to 1b, the Display Engine (DE) will start comparing the display pipe timing generator current scan line value (current scan line) with the start scan line value (current scan line >= start scan line) and the end scan line value (current scan line <= end scan line) to decide if the the pipe scan line is inside or outside the scan line window of interest. DE will wait until the current scan line is either outside (Inclusive mode) or inside (Exclusive mode) the scan line window, then trigger a scan line event and stop any further comparing.

The scan line event can cause display to send a scan line compare response to the render command streamer, used for releasing a MI_WAIT_FOR_EVENT on scan line window, if unmasked in the DERRMR mask register 0x44050. The scan line event can also cause display to generate a scan line compare interrupt, if unmasked and enabled in the DEIMR and DEIER display engine interrupt registers.

The end scan line value must be greater than or equal to the start scan line value.

The value programmed should be the desired value – 1, so for scan line 0, the value programmed is vertical

total, and for scan line 1, the value programmed is 0.

In interlaced display timings, the current scan line is the the current line of the current interlaced field.

Notes for command streamer programming:

- Either MMIO or a MI_LOAD_REGISTER_IMM command can be used to unmask the scan line render response 0x44050.

Bit	Description
31	**Initiate_Compare** Default Value: 0b This field initiates the scan line compare. When this message is received with this bit set to 1b, the display engine will do one complete comparison cycle, trigger a scan line event, then stop comparing. A compare can not be cancelled by writing with this bit set to 0b.

Value	Name	Description	Project
0b	Do nothing	Do nothing	All
1b	Initiate	Initiate the scan line compare.	All

Bit	Description
30	**Inclusive_Exclusive_Select** Default Value: 0b This field selects whether the scan line compare is done in inclusive mode, where display triggers the scan line event when outside the scan line window, or inclusive mode, where display triggers when inside the window.

Value	Name	Description	Project
0b	Exclusive	Exclusive mode: trigger scan line event when inside the scan line window	All
1b	Inclusive	Inclusive mode: trigger scan line event when outside the scan line window	All

Bit	Description
29	**Reserved** Project: All Format:
28:16	**Start_Scan_Line** Project: All Format: This field specifies the starting scan line number of the scan line window.
15	**Load_Source** Project: All Default Value: 0b This bit indicates if the source of the load scan lines is CS or BCS so display can send the scan line event to the appropriate destination.

Value	Name	Description	Project
0b	CS	Source is CS	All
1b	BCS	Source is BCS	All

MSG_LOAD_SL—Load Scan Lines Message	
14:13	**Reserved** Project: All Format:
12:0	**End_Scan_Line** Project: All Format: This field specifies the ending scan line number of the scan line window.

Revision History

Revision Number	Description	Date
1.0	First 2011 OpenSource edition	May 2011

§§

Intel® OpenSource HD Graphics Programmer's Reference Manual (PRM)

Volume 2 Part 1: 3D/Media – 3D Pipeline (SandyBridge)

For the 2011 Intel Core Processor Family

May 2011

Revision 1.0

NOTICE:

This document contains information on products in the design phase of development, and Intel reserves the right to add or remove product features at any time, with or without changes to this open source documentation.

Doc Ref #: IHD-OS-V2 Pt1 – 05 11

Creative Commons License

You are free to Share — to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

No Derivative Works. You may not alter, transform, or build upon this work.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The SandyBridge chipset family, Havendale/Auburndale chipset family, Intel® 965 Express Chipset Family, Intel® G35 Express Chipset, and Intel® 965GMx Chipset Mobile Family Graphics Controller may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel® sales office or your distributor to obtain the latest specifications and before placing your product order. I2C is a two-wire communications bus/protocol developed by Philips. SMBus is a subset of the I2C bus/protocol and was developed by Intel®. Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

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

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

Copyright © 2011, Intel Corporation. All rights reserved.

Contents

1. 3D Pipeline 7
1.1 Introduction 7
1.2 3D Pipeline Overview 7
1.2.1 3D Pipeline Stages 8
1.3 3D Primitives Overview 8
1.4 3D Pipeline State Overview 17
1.4.1 3D State Model 17
1.4.2 3DSTATE_CC_STATE_POINTERS [DevSNB] 18
1.4.3 3DSTATE_BINDING_TABLE_POINTERS 20
1.4.4 3DSTATE_SAMPLER_STATE_POINTERS [DevSNB] 22
1.4.5 3DSTATE_VIEWPORT_STATE_POINTERS [DevSNB+] 24
1.4.6 3DSTATE_SCISSOR_STATE_POINTERS [DevSNB+] 26
1.4.7 3DSTATE_URB [DevSNB] 27
1.4.8 Gather Constants 29
1.5 Vertex Data Overview 29
1.5.1 Vertex URB Entry (VUE) Formats 30
1.5.2 Vertex Positions 39
1.6 3D Pipeline Stage Overview 41
1.6.1 Generic 3D FF Unit Block Diagram 41
1.6.2 Common 3D FF Unit Functions 42
1.6.3 Thread Initiation Management 44
1.6.4 Thread Request Generation 47
1.6.5 Thread Output Handling 54
1.6.6 VUE Readback 56
1.6.7 Statistics Gathering [DevSNB] 56
1.7 Synchronization of the 3D Pipeline 59
1.7.1 Top-of-Pipe Synchronization 59
1.7.2 End-of-Pipe Synchronization 59
1.7.3 Synchronization Actions 59
1.7.4 PIPE_CONTROL Command 60
2. Vertex Fetch (VF) Stage 74
2.1 Vertex Fetch (VF) Stage Overview 74
2.1.1 Input Assembly 74
2.1.2 Vertex Cache 75
2.1.3 Input Data: Push Model vs. Pull Model 75
2.1.4 Generated IDs 75
2.2 Index Buffer (IB) 76
2.2.1 3DSTATE_INDEX_BUFFER [DevSNB+] 77
2.2.2 Index Buffer Access 81
2.3 Vertex Buffers (VBs) 82
2.3.1 3DSTATE_VERTEX_BUFFERS 82
2.3.2 VERTEX_BUFFER_STATE Structure 83
2.3.3 VERTEXDATA Buffers – SEQUENTIAL Access 89
2.3.4 VERTEXDATA Buffers – RANDOM Access 90
2.3.5 INSTANCEDATA Buffers 91
2.4 Input Vertex Definition 91
2.4.1 3DSTATE_VERTEX_ELEMENTS 91
2.4.2 VERTEX_ELEMENT_STATE Structure 93
2.4.3 Vertex Element Data Path 98

2.5 3D Primitive Processing 99
2.5.1 3DPRIMITIVE Command [DevSNB] 99
2.5.2 Functional Overview 106
2.5.3 CommandInit 107
2.5.4 InstanceLoop 107
2.5.5 VertexLoop 107
2.5.6 VertexIndexGeneration 108
2.5.7 TerminatePrimitive 108
2.5.8 VertexCacheLookup 109
2.5.9 VertexElementLoop 109
2.5.10 SourceElementFetch 109
2.5.11 FormatConversion 110
2.5.12 DestinationFormatSelection 115
2.5.13 PrimitiveInfoGeneration 115
2.5.14 URBWrite 117
2.5.15 OutputBufferedVertex 117
2.6 Dangling Vertex Removal 117
2.7 Other Vertex Fetch Functionality 118
2.7.1 Statistics Gathering 118
3. Vertex Shader (VS) Stage 119
3.1 VS Stage Overview 119
3.1.1 Vertex Caching 119
3.2 VS Stage Input 120
3.2.1 State 121
3.2.2 Input Vertices 140
3.3 SIMD4x2 VS Thread Request Generation 140
3.3.1 Thread Payload 141
3.4 SIMD4x2 VS Thread Execution 143
3.4.1 Vertex Output 143
3.4.2 Thread Termination 143
3.5 Primitive Output 144
3.6 Other VS Functions 144
3.6.1 Statistics Gathering 144
4. Geometry Shader (GS) Stage 145
4.1 GS Stage Overview 145
4.2 GS Stage Input 145
4.2.1 State 145
4.3 Object Staging 163
4.4 GS Thread Request Generation 163
4.4.1 Object Vertex Ordering [DevSNB] 163
4.4.2 GS Thread Payload [DevSNB] 167
4.5 GS Thread Execution 170
4.5.1 GS Shader Programming Notes [DevSNB{WA}] 171
4.5.2 Vertex Output [DevSNB] 171
4.5.3 Stream Output 172
4.5.4 Thread Termination 173
4.6 Vertex Header Readback [DevSNB] 174
4.7 Primitive Output 174
4.8 Other Functionality 174
4.8.1 Statistics Gathering 174
5. Clip Stage 176
5.1 CLIP Stage Overview 176
5.1.1 Clip Stage – General-Purpose Processing 176

5.1.2 Clip Stage – 3D Clipping 176
5.1.3 [DevSNB+] Fixed Function Clipper 177
5.2 Concepts 177
5.2.1 The Clip Volume 177
5.2.2 User-Specified Clipping 179
5.2.3 Negative-W Clipping Errata 180
5.2.4 Guard Band 181
5.2.5 Vertex-Based Clip Testing and Considerations 184
5.2.6 3D Clipping 187
5.3 CLIP Stage Input 187
5.3.1 State 187
5.4 VertexClipTest Function 195
5.5 Object Staging 202
5.5.1 Partial Object Removal 202
5.5.2 ClipDetermination Function 203
5.5.3 ClipMode 206
5.6 Object Pass-Through 208
5.7 Primitive Output 210
5.8 Other Functionality 210
5.8.1 Statistics Gathering 210
6. Strips and Fans (SF) Stage 212
6.1 Overview 212
6.1.1 Inputs from CLIP 212
6.1.2 Attribute Setup/Interpolation Process 213
6.1.3 Outputs to WM 213
6.2 Primitive Assembly 214
6.2.1 Point List Decomposition 217
6.2.2 Line List Decomposition 218
6.2.3 Line Strip Decomposition 219
6.2.4 Triangle List Decomposition 221
6.2.5 Triangle Strip Decomposition 222
6.2.6 Triangle Fan Decomposition 223
6.2.7 Polygon Decomposition 224
6.2.8 Rectangle List Decomposition 224
6.3 Object Setup 225
6.3.1 Invalid Position Culling (Pre/Post-Transform) 225
6.3.2 Viewport Transformation 225
6.3.3 Destination Origin Bias 226
6.3.4 Point Rasterization Rule Adjustment 226
6.3.5 Drawing Rectangle Offset Application 228
6.3.6 Point Width Application 233
6.3.7 Rectangle Completion 233
6.3.8 Vertex X,Y Clamping and Quantization 234
6.3.9 Degenerate Object Culling 235
6.3.10 Triangle Orientation (Face) Culling 235
6.3.11 Scissor Rectangle Clipping 236
6.3.12 Line Rasterization 237
6.4 SF Pipeline State Summary 245
6.4.1 3DSTATE_SF [DevSNB+] 245
6.4.2 SF_VIEWPORT [DevSNB] 262
6.4.3 SCISSOR_RECT [DevSNB+] 263
6.5 Attribute Interpolation Setup [DevSNB+] 264
6.5.1 Attribute Swizzling 264

6.5.2 Interpolation Modes 265
6.5.3 Point Sprites 265
6.6 Depth Offset [DevSNB+] 266
6.7 Other SF Functions 266
6.7.1 Statistics Gathering 266
7. Windower (WM) Stage 267
7.1 Overview 267
7.1.1 Inputs from SF to WM 267
7.2 Windower Pipelined State 268
7.2.1 3DSTATE_WM 268
7.2.2 3DSTATE_CONSTANT_PS [DevSNB] 285
7.2.3 3DSTATE_SAMPLE_MASK [DevSNB+] 289
7.3 Rasterization 290
7.3.1 Drawing Rectangle Clipping 290
7.3.2 Line Rasterization 291
7.3.3 Polygon (Triangle and Rectangle) Rasterization 296
7.4 Multisampling [DevSNB+] 300
7.4.1 Multisample Modes/State 300
7.4.2 3DSTATE_MULTISAMPLE [DevSNB+] 305
7.5 Early Depth/Stencil Processing 309
7.5.1 Depth Offset 310
7.5.2 Early Depth Test/Stencil Test/Write 311
7.5.3 Hierarchical Depth Buffer 312
7.5.4 Separate Stencil Buffer 316
7.5.5 Depth/Stencil Buffer State 316
7.6 Barycentric Attribute Interpolation [DevSNB+] 333
7.7 Pixel Shader Thread Generation 333
7.7.1 Pixel Grouping (Dispatch Size) Control 334
7.7.2 Multisampling Effects on Pixel Shader Dispatch [DevSNB+] 336
7.7.3 PS Thread Payload for Normal Dispatch 339
7.8 Other WM Functions 353
7.8.1 Statistics Gathering 353
8. Color Calculator (Output Merger) 354
8.1.1 Alpha Coverage [DevSNB+] 356
8.1.2 Alpha Test 357
8.1.3 Depth Coordinate Offset 358
8.1.4 Stencil Test 359
8.1.5 Depth Test 359
8.1.6 Pre-Blend Color Clamping 360
8.1.7 Color Buffer Blending 361
8.1.8 Post-Blend Color Clamping 364
8.1.9 Color Quantization 364
8.1.10 Dithering 364
8.1.11 Logic Ops 365
8.1.12 Buffer Update 366
8.2 Pixel Pipeline State Summary 368
8.2.1 COLOR_CALC_STATE 368
8.2.2 DEPTH_STENCIL_STATE [DevSNB+] 370
8.2.3 BLEND_STATE [DevSNB+] 375
8.2.4 CC_VIEWPORT 385
8.3 Other Pixel Pipeline Functions 385
8.3.1 Statistics Gathering 385

1. 3D Pipeline

1.1 Introduction

This section covers the programming details for the 3D fixed functions.

1.2 3D Pipeline Overview

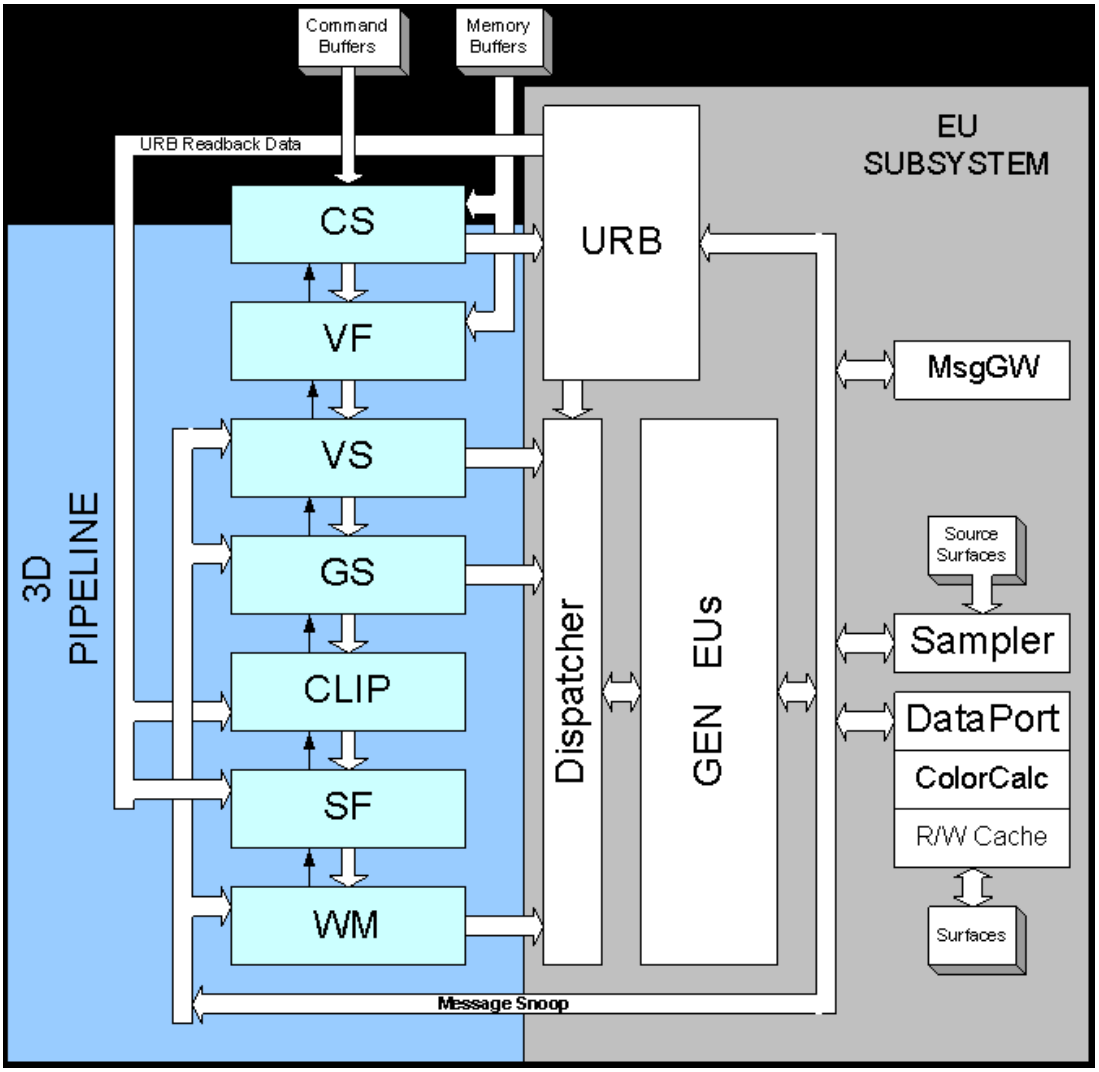


Figure 1 3D Pipeline Diagram [DevSNB]

1.2.1 3D Pipeline Stages

The following table lists the various stages of the 3D pipeline and describes their major functions.

Pipeline Stage	Functions Performed
Command Stream (CS)	The Command Stream stage is responsible for managing the 3D pipeline and passing commands down the pipeline. In addition, the CS unit reads "constant data" from memory buffers and places it in the URB. Note that the CS stage is shared between the 3D and Media pipelines.
Vertex Fetch (VF)	The Vertex Fetch stage, in response to 3D Primitive Processing commands, is responsible for reading vertex data from memory, reformatting it, and writing the results into Vertex URB Entries. It then outputs primitives by passing references to the VUEs down the pipeline.
Vertex Shader (VS)	The Vertex Shader stage is responsible for processing (shading) incoming vertices by passing them to VS threads.
Geometry Shader (GS)	The Geometry Shader stage is responsible for processing incoming objects by passing each object's vertices to a GS thread.
Clipper (CLIP)	The Clipper stage performs clip test on incoming objects and, if required, clips objects via fixed-function hardware [DevGT+].
Strip/Fan (SF)	The Strip/Fan stage performs object setup via use of fixed-function hardware [DevSNB]+..
Windower/Masker (WM)	The Windower/Masker performs object rasterization and spawns WM thread (aka PS thread) to process (shade) the object pixels.

1.3 3D Primitives Overview

The 3DPRIMITIVE command (defined in the *VF Stage* chapter) is used to submit 3D primitives to be processed by the 3D pipeline. Typically the processing results in the rendering of pixel data into the render targets, but this is not required.

> *Terminology Note*: There is considerable confusion surrounding the term 'primitive', e.g., is a triangle strip a 'primitive', or is a triangle within a triangle strip a 'primitive'? In this spec, we will try to avoid ambiguity by using the term 'object' to represent the basic shapes (point, line, triangle), and 'topology' to represent input geometry (strips, lists, etc.). Unfortunately, terms like '3DPRIMITIVE' must remain for legacy reasons.

The following table describes the basic primitive topology types supported in the 3D pipeline.

Notes:

- There are several variants of the basic topologies. These have been introduced to allow slight variations in behavior without requiring a state change.
- Number of vertices:
  - **Dangling Vertices**: Topologies have an "expected" number of vertices in order to form complete objects within the topologies (e.g., LINELIST is expected to have an even number of vertices). The actual number of vertices specified in the 3DPRIMITIVE command, and as output from the GS unit, is allowed to deviate from this expected number --- in which case any "dangling" vertices are discarded. The removal of dangling vertices is initially performed in the VF unit. In order to filter out dangling vertices emitted by GS threads, the CLIP unit also performs dangling-vertex removal at its input. However, the CLIP unit is required to output the expected number.

Table 1. 3D Primitive Topology Types

3D Primitive Topology Type (ordered alphabetically)	Description
LINELIST	A list of independent line objects (2 vertices per line). **Programming Restrictions:** Normal usage expects a multiple of 2 vertices, though incomplete objects are silently ignored.
LINELIST_ADJ	A list of independent line objects with adjacency information (4 vertices per line). **Programming Restrictions:** Normal usage expects a multiple of 4 vertices, though incomplete objects are silently ignored. Not valid as output from GS thread.
LINELOOP	Similar to a 3DPRIM_LINESTRIP, though the last vertex is connected back to the initial vertex via a line object. The LINELOOP topology is converted to LINESTRIP topology at the beginning of the 3D pipeline. **Programming Restrictions:** Normal usage expects at least 2 vertices, though incomplete objects are silently ignored. (The 2-vertex case is required by OGL). Not valid after the GS stage (i.e., must be converted by a GS thread to some other primitive type).

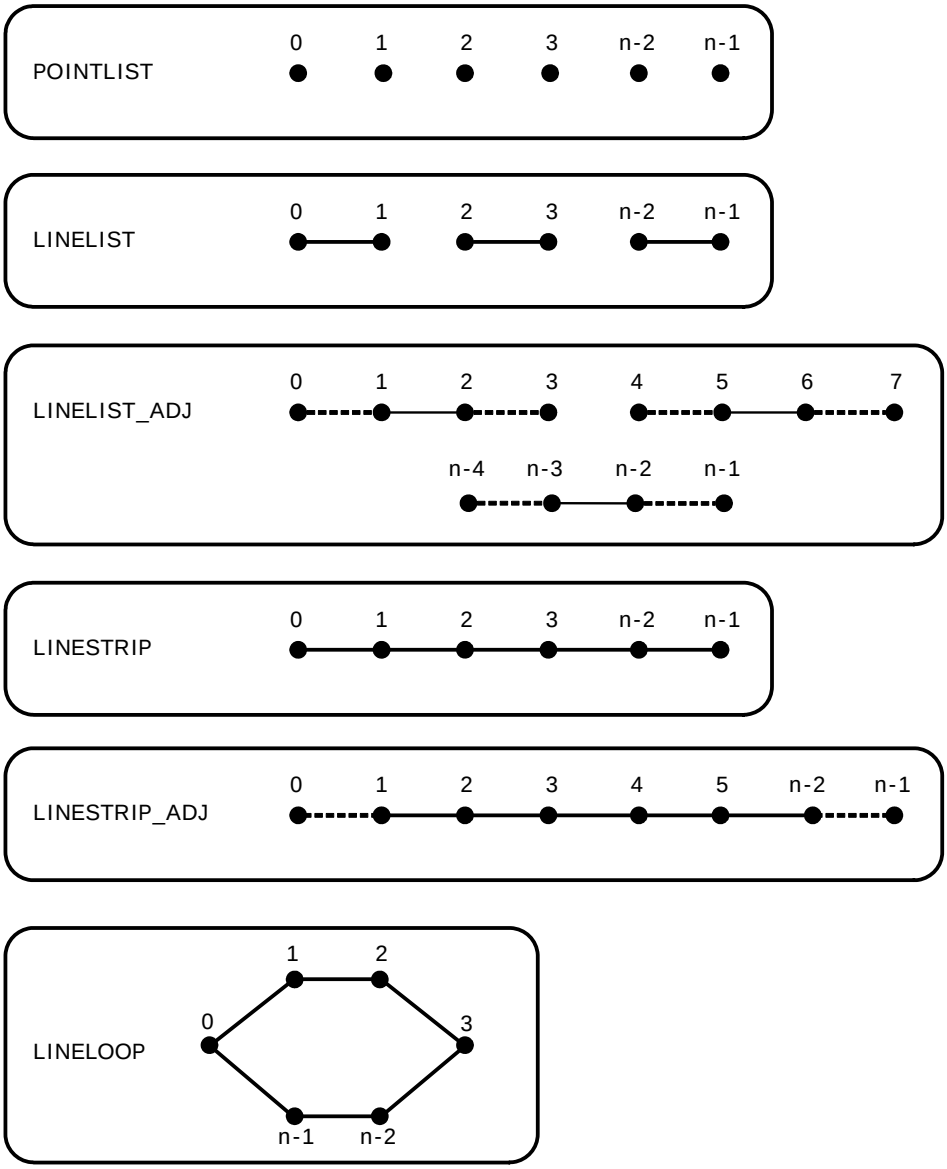
3D Primitive Topology Type (ordered alphabetically)	Description
LINESTRIP	A list of vertices connected such that, after the first vertex, each additional vertex is associated with the previous vertex to define a connected line object. **Programming Restrictions:** Normal usage expects at least 2 vertices, though incomplete objects are silently ignored.
LINESTRIP_ADJ	A list of vertices connected such that, after the first vertex, each additional vertex is associated with the previous vertex to define connected line object. The first and last segments are adjacent–only vertices. **Programming Restrictions:** Normal usage expects at least 4 vertices, though incomplete objects are silently ignored. Not valid as output from GS thread.
LINESTRIP_BF	Similar to LINESTRIP, except treated as “backfacing’ during rasterization (stencil test). This can be used to support “line” polygon fill mode when two-sided stencil is enabled.
LINESTRIP_CONT	Similar to LINESTRIP, except LineStipple (if enabled) is continued (vs. reset) at the start of the primitive topology. This can be used to support line stipple when the API-provided primitive is split across multiple tolopologies.
LINESTRIP_CONT_BF	Combination of LINESTRIP_BF and LINESTRIP_CONT variations.
POINTLIST	A list of point objects (1 vertex per point).
POINTLIST_BF	Similar to POINTLIST, except treated as “backfacing’ during rasterization (stencil test). This can be used to support “point” polygon fill mode when two-sided stencil is enabled.
POLYGON	Similar to TRIFAN, though the first vertex always provides the “flat-shaded” values (vs. this being programmable through state). **Programming Restrictions:** Normal usage expects at least 3 vertices, though incomplete objects are silently ignored.

3D Primitive Topology Type (ordered alphabetically)	Description
QUADLIST	A list of independent quad objects (4 vertices per quad). [DevSNB+]: The QUADLIST topology is converted to POLYGON topology at the beginning of the 3D pipeline. **Programming Restrictions:** Normal usage expects a multiple of 4 vertices, though incomplete objects are silently ignored.
QUADSTRIP	A list of vertices connected such that, after the first two vertices, each additional pair of vertices are associated with the previous two vertices to define a connected quad object. **Programming Restrictions:** Normal usage expects an even number (4 or greater) of vertices, though incomplete objects are silently ignored.
RECTLIST	A list of independent rectangles, where only 3 vertices are provided per rectangle object, with the fourth vertex implied by the definition of a rectangle. V0=LowerRight, V1=LowerLeft, V2=UpperLeft. Implied V3 = V0-V1+V2. **Programming Restrictions:** Normal usage expects a multiple of 3 vertices, though incomplete objects are silently ignored. The RECTLIST primitive is supported specifically for 2D operations (e.g., BLTs and “stretch” BLTs) and not as a general 3D primitive. Due to this, a number of restrictions apply to the use of RECTLIST: Must utilize “screen space” coordinates (VPOS_SCREENSPACE) when the primitive reaches the CLIP stage. The W component of position must be 1.0 for all vertices. The 3 vertices of each object should specify a screen-aligned rectangle (after the implied vertex is computed). Clipping: Must not require clipping or rely on the CLIP unit’s ClipTest logic to determine if clipping is required. Either the CLIP unit should be DISABLED, or the CLIP unit’s Clip Mode should be set to a value other than CLIPMODE_NORMAL. Viewport Mapping must be DISABLED (as is typical with the use of screen-space coordinates).
TRIFAN	Triangle objects arranged in a fan (or polygon). The initial vertex is maintained as a common vertex. After the second vertex, each additional vertex is associated with the previous vertex and the common vertex to define a connected triangle object . **Programming Restrictions:** Normal usage expects at least 3 vertices, though incomplete objects are silently ignored.

3D Primitive Topology Type (ordered alphabetically)	Description
TRIFAN_NOSTIPPLE	Similar to TRIFAN, but poylgon stipple is not applied (even if enabled). This can be used to support “point” polygon fill mode, under the combination of the following conditions: (a) when the frontfacing and backfacing polygon fill modes are different (so the final fill mode is not known to the driver), (b) one of the fill modes is “point” and the other is “solid”, (c) point mode is being emulated by converting the point into a trifan, (d) polygon stipple is enabled. In this case, polygon stipple should not be applied to the points-emulated-as-trifans.
TRILIST	A list of independent triangle objects (3 vertices per triangle). **Programming Restrictions:** Normal usage expects a multiple of 3 vertices, though incomplete objects are silently ignored.
TRILIST_ADJ	A list of independent triangle objects with adjacency information (6 vertices per triangle). **Programming Restrictions:** Normal usage expects a multiple of 6 vertices, though incomplete objects are silently ignored. Not valid as output from GS thread.
TRISTRIP	A list of vertices connected such that, after the first two vertices, each additional vertex is associated with the last two vertices to define a connected triangle object. **Programming Restrictions:** Normal usage expects at least 3 vertices, though incomplete objects are silently ignored.
TRISTRIP_ADJ	A list of vertices where the even-numbered (including 0th) vertices are connected such that, after the first two vertex pairs, each additional even-numbered vertex is associated with the last two even-numbered vertices to define a connected triangle object. The odd-numbered vertices are adjacent-only vertices. **Programming Restrictions:** Normal usage expects at least 6 vertices, though incomplete objects are silently ignored. Not valid as output from GS thread.

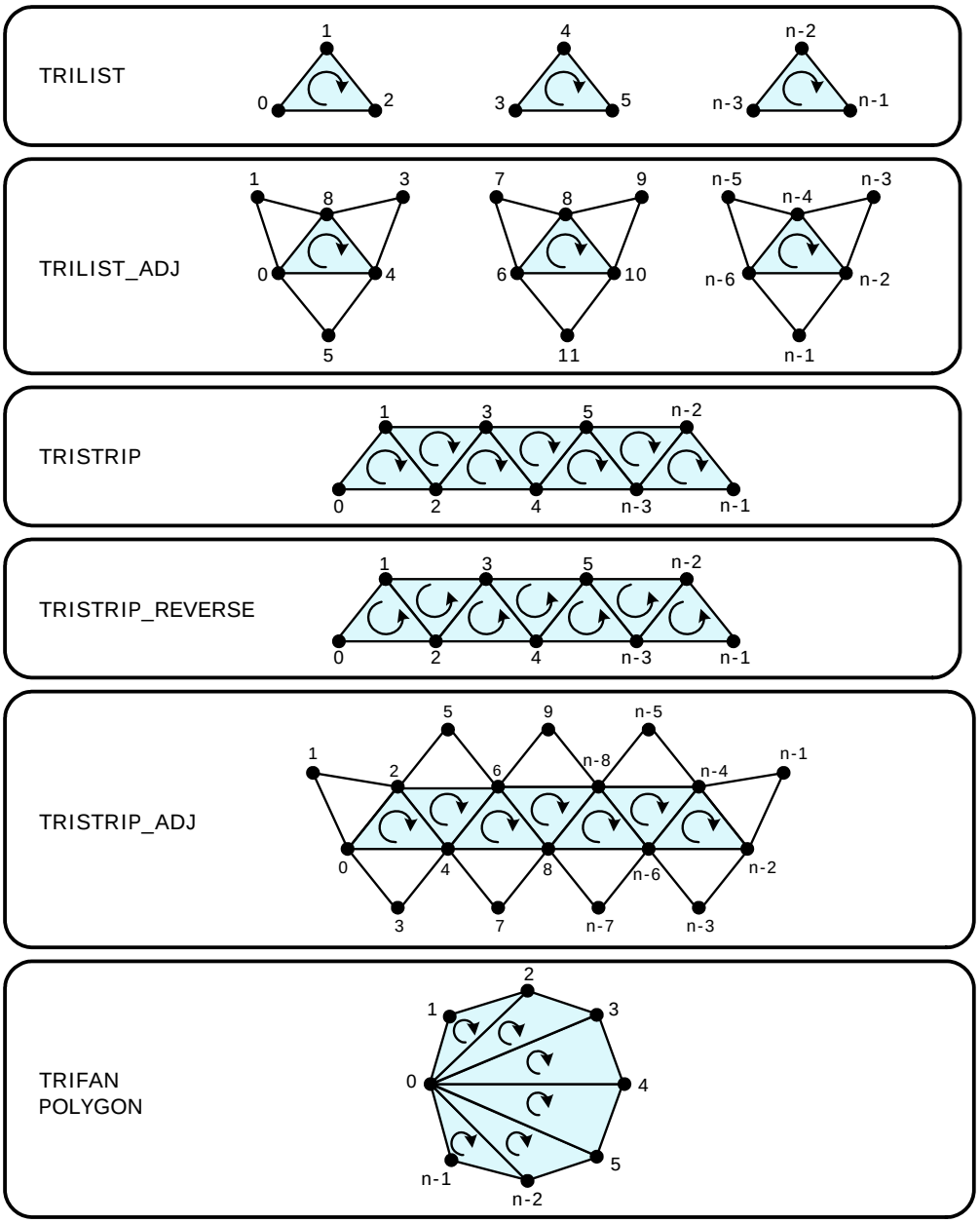
3D Primitive Topology Type (ordered alphabetically)	Description
TRISTRIP_REVERSE	Similar to TRISTRIP, though the sense of orientation (winding order) is reversed – this allows SW to break long tristrips into smaller pieces and still maintain correct face orientations.

The following diagrams illustrate the basic 3D primitive topologies. (Variants are not shown if they have the same definition with respect to the information provided in the diagrams).

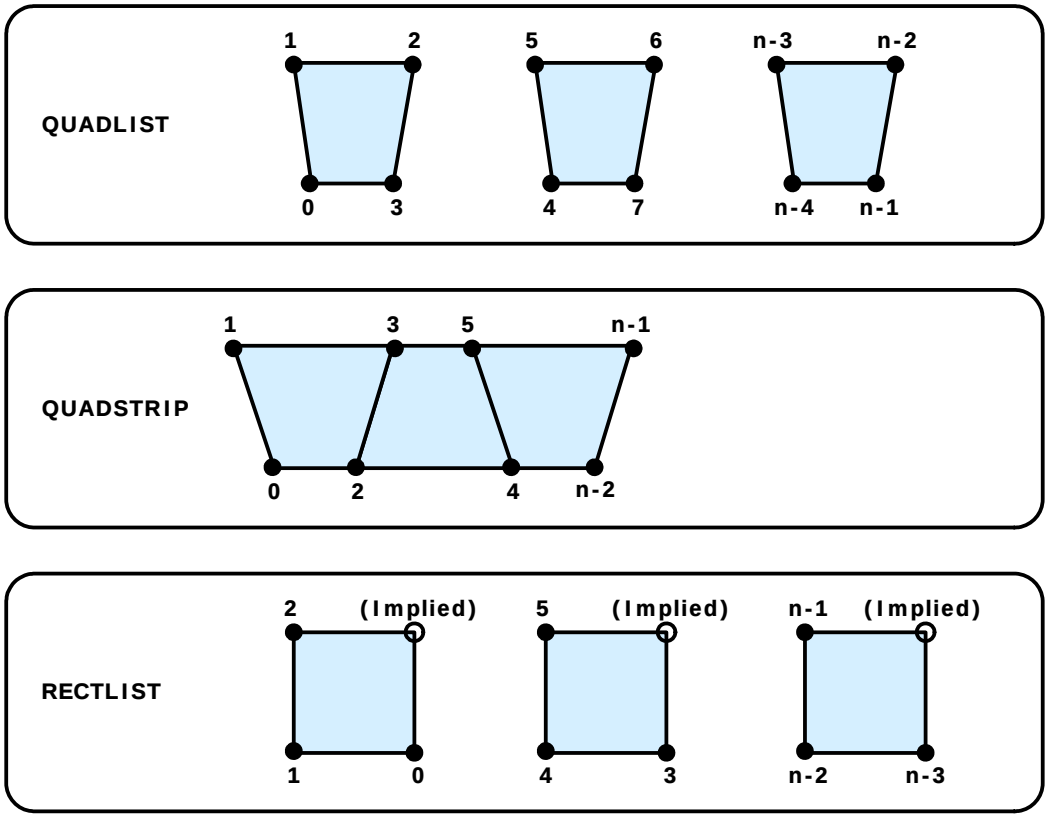


B6815-01

A note on the arrows you see below: These arrows are intended to show the vertex ordering of triangles that are to be considered having “clockwise” winding order in screen space. Effectively, the arrows show the order in which vertices are used in the cross-product (area, determinant) computation. Note that for TRISTRIP, this requires that either the order of odd-numbered triangles be reversed in the cross-product or the sign of the result of the normally-ordered cross-product be flipped (these are identical operations).



B6816-01


QUADLIST
1
2
0
3
5
6
4
7
n-3
n-2
n-4
n-1
QUADSTRIP
1
3
5
n-1
0
2
4
n-2
RECTLIST
2
(Implied)
1
0
5
(Implied)
4
3
n-1
(Implied)
n-2
n-3

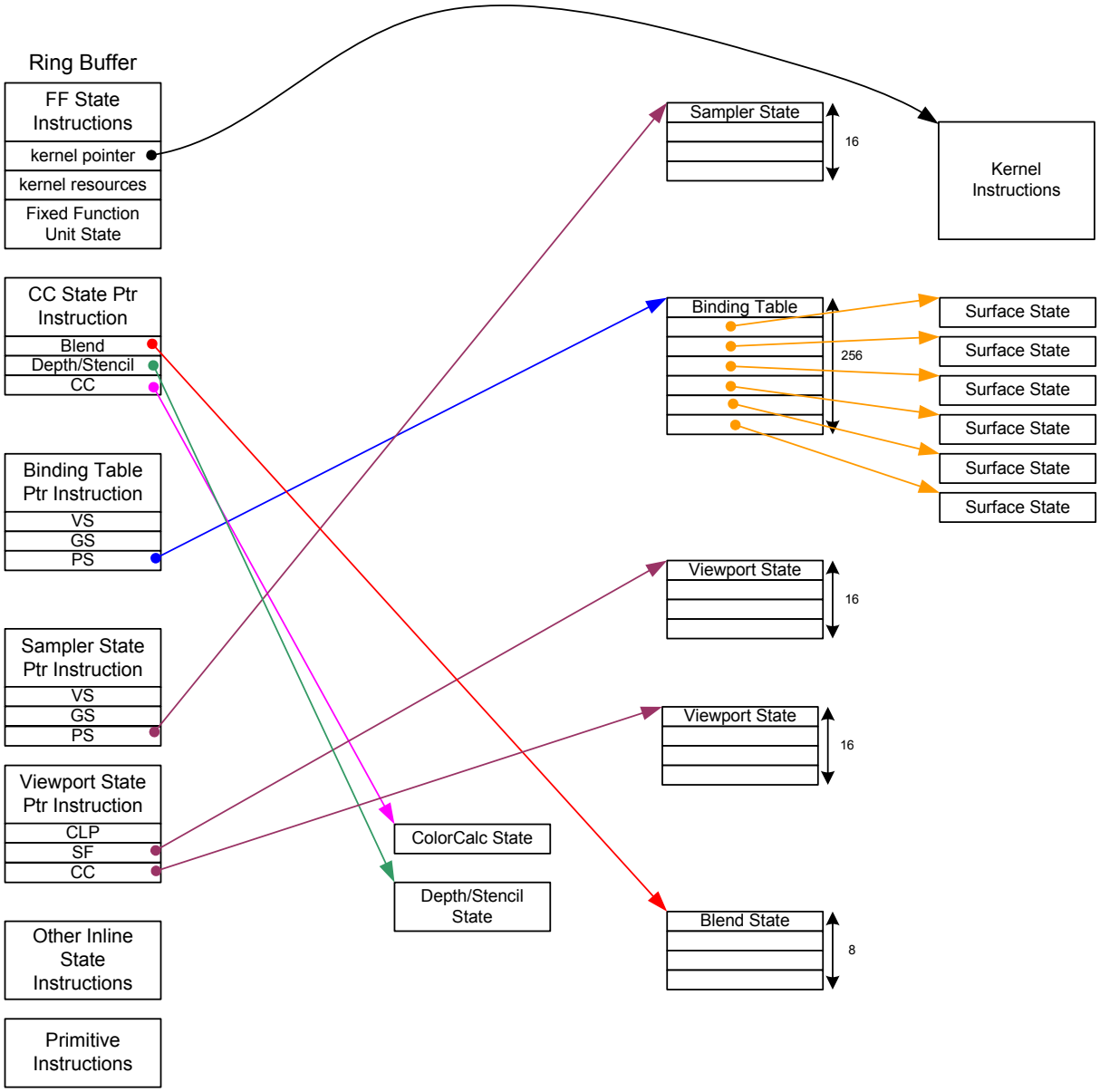
B6818-01

1.4 3D Pipeline State Overview

1.4.1 3D State Model

1.4.1.1 3D State Model [DevSNB+]

The locations of the sampler state and viewport state pointers have been moved from the state descriptors to the ring buffer. In addition, the state for the fixed function pipeline has been moved from indirect state descriptors to inline commands. The color calculator state has been repartitioned.



1.4.2 3DSTATE_CC_STATE_POINTERS [DevSNB]

3DSTATE_CC_STATE_POINTERS			
Project:	[DevSNB]	**Length Bias:**	2
The 3DSTATE_CC_STATE_POINTERS command is used to set up the pointers to the color calculator state.			

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Eh 3DSTATE_CC_STATE_POINTERS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:6	**Pointer to BLEND_STATE** Project: All Address: DynamicStateOffset[31:6] Surface Type: BLEND_STATE*8 Specifies the 64-byte aligned offset of the BLEND_STATE. This offset is relative to the **Dynamic State Base Address**.
	5:1	**Reserved** Project: All Format: MBZ
	0	**BLEND_STATE Change** Project: All Format: Enable FormatDesc This bit, if set, indicates that the BLEND_STATE pointer has changed and new state needs to be fetched.

3DSTATE_CC_STATE_POINTERS		
2	31:6	**Pointer to DEPTH_STENCIL_STATE** Project: All Address: DynamicStateOffset [31:6] Surface Type: DEPTH_STENCIL_STATE Specifies the 64-byte aligned offset of the DEPTH_STENCIL_STATE. This offset is relative to the **Dynamic State Base Address**.
	5:1	**Reserved** Project: All Format: MBZ
	0	**DEPTH_STENCIL_STATE Change** Project: All Format: Enable FormatDesc This bit, if set, indicates that the DEPTH_STENCIL_STATE pointer has changed and new state needs to be fetched.
3	31:6	**Pointer to COLOR_CALC_STATE** Project: All Address: DynamicStateOffset[31:6] Surface Type: COLOR_CALC_STATE Specifies the 64-byte aligned offset of the COLOR_CALC_STATE. This offset is relative to the **Dynamic State Base Address**.
	5:1	**Reserved** Project: All Format: MBZ
	0	**COLOR_CALC_STATE Change** Project: All Format: Enable FormatDesc This bit, if set, indicates that the COLOR_CALC_STATE pointer has changed and new state needs to be fetched.

1.4.3 3DSTATE_BINDING_TABLE_POINTERS

1.4.3.1 3DSTATE_BINDING_TABLE_POINTERS [DevSNB]

3DSTATE_BINDING_TABLE_POINTERS			
Project:	[DevSNB]	**Length Bias:**	2
The 3DSTATE_BINDING_TABLE_POINTERS command is used to define the location of fixed functions' BINDING_TABLE_STATE. Only some of the fixed functions utilize binding tables.			

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 01h 3DSTATE_BINDING_TABLE_POINTERS Format: OpCode
	15:13	**Reserved** Project: All Format: MBZ
	12	**PS Binding Table Change** Project: All Format: Enable FormatDesc This bit, if set, indicates that the PS (Windower) Binding Table pointer has changed and new state needs to be fetched.
	11:10	**Reserved** Project: All Format: MBZ

3DSTATE_BINDING_TABLE_POINTERS		
	9	**GS Binding Table Change** Project: All Format: Enable FormatDesc This bit, if set, indicates that the GS Binding Table pointer has changed and new state needs to be fetched.
	8	**VS Binding Table Change** Project: All Format: Enable FormatDesc This bit, if set, indicates that the VS Binding Table pointer has changed and new state needs to be fetched.
	7:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:5	Pointer to VS Binding Table Project: All Address: SurfaceStateOffset[31:5] Surface Type: BINDING_TABLE_STATE*256 Specifies the 32-byte aligned address offset of the VS function's BINDING_TABLE_STATE. This offset is relative to the **Surface State Base Address**. For [DevSNB-B+], bits [31:16] of this field must be set to zero.
	4:0	Reserved Project: All Format: MBZ
2	31:5	Pointer to GS Binding Table Project: All Address: SurfaceStateOffset[31:5] Surface Type: BINDING_TABLE_STATE*256 Specifies the 32-byte aligned address offset of the GS function's BINDING_TABLE_STATE. This offset is relative to the **Surface State Base Address**. For [DevSNB-B+], bits [31:16] of this field must be set to zero.
	4:0	Reserved Project: All Format: MBZ

3DSTATE_BINDING_TABLE_POINTERS		
3	31:5	Pointer to PS Binding Table Project: All Address: SurfaceStateOffset[31:5] Surface Type: BINDING_TABLE_STATE*256 Specifies the 32-byte aligned address offset of the PS (Windower) function's BINDING_TABLE_STATE. This offset is relative to the **Surface State Base Address**. For [DevSNB-B+], bits [31:16] of this field must be set to zero.
	4:0	Reserved Project: All Format: MBZ

1.4.4 3DSTATE_SAMPLER_STATE_POINTERS [DevSNB]

3DSTATE_SAMPLER_STATE_POINTERS			
Project:	[DevSNB]	**Length Bias:**	2
The 3DSTATE_SAMPLER_STATE_POINTERS command is used to define the location of fixed functions' SAMPLER_STATE table. Only some of the fixed functions utilize sampler state tables.			
DWord	**Bit**	**Description**	
0	31:29	Command Type Default Value: 3h GFXPIPE Format: OpCode	
	28:27	Command SubType Default Value: 3h GFXPIPE_3D Format: OpCode	
	26:24	3D Command Opcode Default Value: 0h 3DSTATE_PIPELINED Format: OpCode	
	23:16	3D Command Sub Opcode Default Value: 02h 3DSTATE_SAMPLER_STATE_POINTERS Format: OpCode	
	15:13	Reserved Project: All Format: MBZ	

3DSTATE_SAMPLER_STATE_POINTERS		
	12	PS Sampler State Change Project: All This bit, if set, indicates that the PS (Windower) Sampler State pointer has changed and new state needs to be fetched.
	11:10	Reserved Project: All Format: MBZ
	9	GS Sampler State Change Project: All This bit, if set, indicates that the GS Sampler State pointer has changed and new state needs to be fetched.
	8	VS Sampler State Change Project: All This bit, if set, indicates that the VS Sampler State pointer has changed and new state needs to be fetched.
	7:0	DWord Length Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:5	Pointer to VS Sampler State Project: All Address: DynamicStateOffset[31:5] Surface Type: SAMPLER_STATE*16 Specifies the 32-byte aligned address offset of the VS function's SAMPLER_STATE table. This offset is relative to the Dynamic State Base Address.
	4:0	Reserved Project: All Format: MBZ

3DSTATE_SAMPLER_STATE_POINTERS		
2	31:5	Pointer to GS Sampler State Project: All Address: DynamicStateOffset[31:5] Surface Type: SAMPLER_STATE*16 Specifies the 32-byte aligned address offset of the GS function's SAMPLER_STATE table. This offset is relative to the Dynamic State Base Address.
	4:0	Reserved Project: All Format: MBZ
3	31:5	Pointer to PS Sampler State Project: All Address: DynamicStateOffset[31:5] Surface Type: SAMPLER_STATE*16 Specifies the 32-byte aligned address offset of the PS (Windower) function's SAMPLER_STATE table. This offset is relative to the Dynamic State Base Address.
	4:0	Reserved Project: All Format: MBZ

1.4.5 3DSTATE_VIEWPORT_STATE_POINTERS [DevSNB+]

3DSTATE_VIEWPORT_STATE_POINTERS			
Project:	[DevSNB+]	**Length Bias:**	2
The 3DSTATE_VIEWPORT_STATE_POINTERS command is used to define the location of fixed functions' viewport state table (CLIP_VIEWPORT, SF_VIEWPORT, or CC_VIEWPORT).			

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

3DSTATE_VIEWPORT_STATE_POINTERS		
	23:16	3D Command Sub Opcode Default Value: 0Dh 3DSTATE_VIEWPORT_STATE_POINTERS Format: OpCode
	15:13	Reserved Project: All Format: MBZ
	12	CC Viewport State Change Project: All This bit, if set, indicates that the CC_VIEWPORT pointer has changed and new state needs to be fetched.
	11	SF Viewport State Change Project: All This bit, if set, indicates that the SF_VIEWPORT pointer has changed and new state needs to be fetched.
	10	CLIP Viewport State Change Project: All This bit, if set, indicates that the CLIP_VIEWPORT pointer has changed and new state needs to be fetched.
	9:8	Reserved Project: All Format: MBZ
	7:0	DWord Length Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:5	Pointer to CLIP_VIEWPORT Project: All Address: DynamicStateOffset[31:5] Surface Type: CLIP_VIEWPORT*16 Specifies the 32-byte aligned address offset of the CLIP_VIEWPORT state. This offset is relative to the Dynamic State Base Address.
	4:0	Reserved Project: All Format: MBZ
2	31:5	Pointer to SF_VIEWPORT Project: All Address: DynamicStateOffset[31:5] Surface Type: SF_VIEWPORT*16 Specifies the 32-byte aligned address offset of the SF_VIEWPORT state. This offset is relative to the Dynamic State Base Address.
	4:0	Reserved Project: All Format: MBZ

3DSTATE_VIEWPORT_STATE_POINTERS		
3	31:5	Pointer to CC_VIEWPORT Project: All Address: DynamicStateOffset[31:5] Surface Type: CC_VIEWPORT*16 Specifies the 32-byte aligned address offset of the CC_VIEWPORT state. This offset is relative to the Dynamic State Base Address.
	4:0	Reserved Project: All Format: MBZ

1.4.6 3DSTATE_SCISSOR_STATE_POINTERS [DevSNB+]

3DSTATE_SCISSOR_STATE_POINTERS			
Project:	[DevSNB+]	**Length Bias:**	2
The 3DSTATE_SCISSOR_STATE_POINTERS command is used to define the location of the indirect SCISSOR_RECT state.			

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

3DSTATE_SCISSOR_STATE_POINTERS		
1	31:5	Pointer to SCISSOR_RECT Project: All Address: DynamicStateOffset[31:5] Surface Type: SCISSOR_RECT Specifies the 32-byte aligned address offset of the SCISSOR_RECT state. This offset is relative to the **Dynamic State Base Address**.
	4:0	Reserved Project: All Format: MBZ

1.4.7 3DSTATE_URB [DevSNB]

The URB for [DevSNB] is partitioned only between the VS and GS units. The following command determines that partitioning within the URB.

Because of a urb corruption caused by allocating a previous gsunit's urb entry to vsunit software is required to send a "GS NULL Fence"(Send URB fence with VS URB size == 1 and GS URB size == 0) plus a dummy DRAW call before any case where VS will be taking over GS URB space.

3DSTATE_URB			
Project:	[DevSNB]	**Length Bias:**	2
DWord	**Bit**	**Description**	
0	31:29	Command Type Default Value: 3h GFXPIPE Format: OpCode	
	28:27	Command SubType Default Value: 3h GFXPIPE_3D Format: OpCode	
	26:24	3D Command Opcode Default Value: 0h 3DSTATE_PIPELINED Format: OpCode	

3DSTATE_URB		
	23:16	3D Command Sub Opcode Default Value: 05h 3DSTATE_URB Format: OpCode
	15:8	Reserved Project: All Format: MBZ
	7:0	DWord Length Default Value: 1h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:24	Reserved Project: All Format: MBZ
	23:16	VS URB Entry Allocation Size Project: All Format: U3 count (of 1024-bit units – 1) Range [0,4] = [1,5] 1024-bit URB rows Specifies the length of each URB entry owned by VS. This field is always used (even if **VS Function Enable** is DISABLED).
	15:0	VS Number of URB Entries Project: All Format: U16 FormatDesc Range [24,256] in multiples of 4 [24, 128] in multiples of 4[DevSNBGT1] Specifies the number of URB entries that are used by VS. This field is always used (even if **VS Function Enable** is DISABLED).
2	31:18	Reserved Project: All Format: MBZ

3DSTATE_URB		
	17:8	**GS Number of URB Entries** Project: All Format: U9 FormatDesc Range [0,256] in multiples of 4 [0, 254] in multiples of 4[DevSNBGT1] Specifies the number of URB entries that are used by GS. Note: This value must be non-zero value if the GS is dispatching threads that require handles.
	7:3	Reserved Project: All Format: MBZ
	2:0	**GS URB Entry Allocation Size** Project: All Format: U3 FormatDesc Range [0,4] = [1,5] 1024-bit URB rows Specifies the size of each URB entry used by the GS.

1.4.8 Gather Constants

The compiler does some optimizations of constant usage and determines which elements of which constants should be packed in which push constant register for optimum shader performance. While this gathering and packing of constant elements into push constant registers optimizes the shader, it cause the driver additional work at draw call time, since the driver must do the gather and packing at draw time. A new cmd 3D_STATE_GATHER_CONSTANT_* is added to offload the gather and packing functions from the driver. There are 5 FF which support push constants (VS, GS, DS, HS, PS) and they all have corresponding gather cmds. The compiler generates a gather table which instructs what elements of what buffers should be pack into the gather buffer. The gather table indexes the BT to get the surface state which points to the constant buffer. The resource streamer fills gather buffer when it executes a 3D_STATE_GATHER_CONSTANT_* cmd. Once the gather buffer has been filled, the Cmd streamer will execute the 3D_STATE_CONSTANT_* to load the push constant into the URB. **Note:** The gather push constants can only be used if the HW generated binding tables are also used.

1.5 Vertex Data Overview

The 3D pipeline FF stages (past VF) receive input 3D primitives as a stream of vertex information packets. (These packets are not directly visible to software). Much of the data associated with a vertex is passed indirectly via a VUE handle. The information provided in vertex packets includes:

- The **URB Handle** of the VUE: This is used by the FF unit to refer to the VUE and perform any required operations on it (e.g., cause it to be read into the thread payload, dereference it, etc.).

- **Primitive Topology Information**: This information is used to identify/delineate primitive topologies in the 3D pipeline. Initially, the VF unit supplies this information, which then passes thru the VS stage unchanged. GS and CLIP threads must supply this information with each vertex they produce (via the URB_WRITE message). If a FF unit directly outputs vertices (that were not generated by a thread they spawned), that FF unit is responsible for providing this information.

    - **PrimType**: The type of topology, as defined by the corresponding field of the 3DPRIMITIVE command.

    - **StartPrim**: TRUE only for the first vertex of a topology.

    - **EndPrim**: TRUE only for the last vertex of a topology.

- (Possibly, depending on FF unit) Data read back from the **Vertex Header** of the VUE.

1.5.1 Vertex URB Entry (VUE) Formats

In general, vertex data is stored in Vertex URB Entries (VUEs) in the URB, processed by CLIP threads, and only referenced by the pipeline stages indirectly via VUE handles. Therefore (for the most part) the contents/format of the vertex data is not exposed to 3D pipeline hardware – the FF units are typically only aware of the handles and sizes of VUEs.

VUEs are written in two ways:

- At the top of the 3D Geometry pipeline, the VF's InputAssembly function creates VUEs and initializes them from data extracted from Vertex Buffers as well as internally-generated data.

- VS, GS, and CLIP threads can compute, format and write new VUEs as thread output.

There are only two points in the 3D FF pipeline where the FF units are exposed to the VUE data. Otherwise the VUE remains opaque to the 3D pipeline hardware.

- Just prior to the CLIP stage, all VUEs are read-back:

    - [**Pre-DevIL**] Readback of the Vertex Header (first 256 bits of the VUE)

    - [**DevIL**] Readback of the Vertex Header (first 512 bits of the VUE)

    - [**DevIL**] Optional readback of User Clip distances if the User Clip Planes are enabled.

    - [**DevSNB+**]: Optional readback of ClipDistance values (up to 8 floats in an aligned 256-bit URB row)

- Just after the CLIP stage, on clip-generated VUEs are read-back:

    - Readback of the Vertex Header (first 256 bits of the VUE)

Software must ensure that any VUEs subject to readback by the 3D pipeline start with a valid Vertex Header. This extends to all VUEs with the following exceptions listed below:

- If the VS function is enabled, the VF-written VUEs are not required to have Vertex Headers, as the VS-incoming vertices are guaranteed to be consumed by the VS (i.e., the VS thread is responsible for overwriting the input vertex data).

- If the GS FF is enabled, neither VF-written VUEs nor VS thread-generated VUEs are required to have Vertex Headers, as the GS will consume all incoming vertices.

- (There is a pathological case where the CLIP state can be programmed to guarantee that all CLIP-incoming vertices are consumed – regardless of the data read back prior to the CLIP stage – and therefore only the CLIP thread-generated vertices would require Vertex Headers).

The folllowing table defines the Vertex Header. The Position fields are described in further detail below.

Table 2. VUE Vertex Header ([Pre-DevIL])

DWord	Bit	Description
D0	31:0	Reserved: MBZ
D1	31:11	Reserved: MBZ
	10:0	**Render Target Array Index.** This value is (eventually) used to index into a specific element of an "array" Render Target. It is read back by the GS unit (for all exiting vertices) and the Clip unit (for all clip-generated vertices), subsequently routed into the PS thread payload, and eventually included in the RTWrite DataPort message header for use by the DataPort shared function. Software is responsible for ensuring this field is zero whenever a programmable index value is <u>not</u> required. When a programmable index value is required software must ensure that the correct 11-bit value is written to this field. Specifically, the kernels must perform a range check of computed index values against [0,2047], and output zero if that range is exceeded. Note that the unmodified "renderTargetArrayIndex" must be maintained in the VUE outside of the Vertex Header. Downstream, the DataPort range-checks the 11-bit index values against the range [**MinimumArrayElement**, **Depth**] state variables (SURFACE_STATE) associated with the specified render target surface. Format: 0-based U11 index value

DWord	Bit	Description
D2	31:0	**Viewport Index.** This value is used to select one of a possible 16 sets of viewport (VP) state parameters in the Clip unit's VertexClipTest function and in the SF unit's ViewportMapping and Scissor functions. The GS unit (even if disabled) will read back this value for all vertices exiting the GS stage and entering the Clip stage. When enabled, the GS unit will range-check the value against [0,**Maximum VPIndex]** (see GS_STATE) and use a value of zero if out-of-range. When disabled, the GS unit instead uses the range [0,15]. After this range-check the values are sent down the pipeline and used in the Clip unit's VertexClipTest function. For vertices passing through the Clip stage, these values will also be sent to the SF unit for use in ViewportMapping and Scissor functions. The Clip unit (if enabled) will read back this value only for vertices generated by CLIP threads. Unlike the GS unit, the Clip unit will not apply any range check and instead just use the lower 4 bits. No hardware clamping is performed on these read-back values – the read-back values will be used unmodified by the SF unit. The CLIP kernel is therefore responsible for performing any required clamping on this value prior to writing the VUE Vertex Header. Software is responsible for ensuring this field is zero whenever a programmable index value is <u>not</u> required. Format: 0-based U32 index value
D3	31:19	Reserved: MBZ
	18:8	**Point Width.** This field specifies the width of POINT objects in screen-space pixels. It is used only for vertices within POINTLIST and POINTLIST_BF primitive topologies, and is ignored for vertices associated with other primitive topologies. This field is read back by both the GS and Clip units. Format: U8.3 pixels
	7:0	**User Clip Codes.** These are 'outside' status bits associated with the vertex element components marked as CullDistance or ClipDistance. The JITTER is required to generate code to compute and pack these bits. If a Cull/ClipDistance value is negative or a NaN value, its corresponding User Clip Code bit should be set. Up to eight values/bits are supported. The CLIP unit supports the **UserClipFlag ClipTest Enable Bitmask** (CLIP_STATE) which is applied to this field before being used in ClipTest. This field is read back only by the GS unit. This field is ignored for CLIP thread-generated vertices, as this information is only relevant to CLIP input vertices. Format: BITMASK8

DWord	Bit	Description
D4	31:0	**Vertex Position X Coordinate.** If this is a PREMAPPED vertex, this field contains the X component of the vertex's screen space position. If this is an UNMAPPED vertex, this field contains the X component of the vertex's NDC space position (i.e., the clip space X component divided by the clip space W component). Format: FLOAT32
D5	31:0	**Vertex Position Y Coordinate.** If this is a PREMAPPED vertex, this field contains the Y component of the vertex's screen space position. If this is an UNMAPPED vertex, this field contains the Y component of the vertex's NDC space position (i.e., the clip space Y component divided by the clip space W component). Format: FLOAT32
D6	31:0	**Vertex Position Z Coordinate.** If this is a PREMAPPED vertex, this field contains the Z component of the vertex's screen space position. If this is an UNMAPPED vertex, this field contains the Z component of the vertex's NDC space position (i.e., the clip space Z component divided by the clip space W component). Format: FLOAT32
D7	31:0	**Vertex Position RHW Coordinate.** This field contains the reciprocal of the vertex's clip space W coordinate. Format: FLOAT32
(D8-Dn)	31:0	**(Remainder of Vertex Elements).** While DWords D0-D7 are exposed to the device (i.e., read back by FF units), DWords D8-Dn of vertices written (by threads) are opaque to the device. Software is free to format/use these DWords as desired. The absolute maximum size limit on this data is specified via a maximum limit on the amount of data that can be read from a VUE (including the Vertex Header) (**Vertex Entry URB Read Length** has a maximum value of 63 256-bit units). Therefore the Remainder of Vertex Elements has an absolute maximum size of 62 256-bit units. Of course the actual allocated size of the VUE can and will limit the amount of data in a VUE.

Table 3 VUE Vertex Header ([DevIL])

DWord	Bit	Description
D0	31:0	Reserved: MBZ
D1	31:11	Reserved: MBZ
	10:0	**Render Target Array Index.** This value is (eventually) used to index into a specific element of an "array" Render Target. It is read back by the GS unit (for all exiting vertices) and the Clip unit (for all clip-generated vertices), subsequently routed into the PS thread payload, and eventually included in the RTWrite DataPort message header for use by the DataPort shared function. Software is responsible for ensuring this field is zero whenever a programmable index value is not required. When a programmable index value is required software must ensure that the correct 11-bit value is written to this field. Specifically, the kernels must perform a range check of computed index values against [0,2047], and output zero if that range is exceeded. Note that the unmodified "renderTargetArrayIndex" must be maintained in the VUE outside of the Vertex Header. Downstream, the DataPort range-checks the 11-bit index values against the range [**MinimumArrayElement**, **Depth**] state variables (SURFACE_STATE) associated with the specified render target surface. Format: 0-based U11 index value
D2	31:0	**Viewport Index.** This value is used to select one of a possible 16 sets of viewport (VP) state parameters in the Clip unit's VertexClipTest function and in the SF unit's ViewportMapping and Scissor functions. The GS unit (even if disabled) will read back this value for all vertices exiting the GS stage and entering the Clip stage. When enabled, the GS unit will range-check the value against [0,**Maximum VPIndex]** (see GS_STATE) and use a value of zero if out-of-range. When disabled, the GS unit instead uses the range [0,15]. After this range-check the values are sent down the pipeline and used in the Clip unit's VertexClipTest function. For vertices passing through the Clip stage, these values will also be sent to the SF unit for use in ViewportMapping and Scissor functions. The Clip unit (if enabled) will read back this value only for vertices generated by CLIP threads. Unlike the GS unit, the Clip unit will not apply any range check and instead just use the lower 4 bits. No hardware clamping is performed on these read-back values – the read-back values will be used unmodified by the SF unit. The CLIP kernel is therefore responsible for performing any required clamping on this value prior to writing the VUE Vertex Header. Software is responsible for ensuring this field is zero whenever a programmable index value is not required. Format: 0-based U32 index value

DWord	Bit	Description
D3	31:19	Reserved: MBZ
	18:8	**Point Width.** This field specifies the width of POINT objects in screen-space pixels. It is used only for vertices within POINTLIST and POINTLIST_BF primitive topologies, and is ignored for vertices associated with other primitive topologies. This field is read back by both the GS and Clip units. Format: U8.3 pixels
	7:0	**User Clip Codes.** These are the sign bits of the vertex element components marked as CullDistance or ClipDistance. The JITTER is required to assemble these sign bits. A negative value (sign bit set) indicates that the vertex is on the "outside" of the corresponding user clip plane. Up to eight sign bits (clip flags) are supported. The CLIP unit supports a mask that is applied to this field before being used in ClipTest. This field is read back only by the GS unit. This field is ignored for CLIP thread-generated vertices, as this information is only relevant to CLIP input vertices. Format: BITMASK8
D4	31:0	**Vertex Position X Coordinate.** If this is a PREMAPPED vertex, this field contains the X component of the vertex's screen space position. If this is an UNMAPPED vertex, this field contains the X component of the vertex's NDC space position (i.e., the clip space X component divided by the clip space W component). Format: FLOAT32
D5	31:0	**Vertex Position Y Coordinate.** If this is a PREMAPPED vertex, this field contains the Y component of the vertex's screen space position. If this is an UNMAPPED vertex, this field contains the Y component of the vertex's NDC space position (i.e., the clip space Y component divided by the clip space W component). Format: FLOAT32
D6	31:0	**Vertex Position Z Coordinate.** If this is a PREMAPPED vertex, this field contains the Z component of the vertex's screen space position. If this is an UNMAPPED vertex, this field contains the Z component of the vertex's NDC space position (i.e., the clip space Z component divided by the clip space W component). Format: FLOAT32

DWord	Bit	Description
D7	31:0	**Vertex Position RHW Coordinate.** This field contains the reciprocal of the vertex's clip space W coordinate.
D8	31:0	**Vertex Position X Coordinate.** This field contains the X component of the vertex's 4D space position. Format: FLOAT32
D9	31:0	**Vertex Position Y Coordinate.** This field contains the Y component of the vertex's 4D space position Format: FLOAT32
D10	31:0	**Vertex Position Z Coordinate.** This field contains the Z component of the vertex's NDC space position Format: FLOAT32
D11	31:0	**Vertex Position W Coordinate.** This field contains the Z component of the vertex's 4D space position Format: FLOAT32
D12	31:0	**User Clip Distance to Plane0.** If the User Clip Plane0 is enabled,This field contains distance from the vertex to the User Clip Plane0 Format: FLOAT32
D13	31:0	**User Clip Distance to Plane1.** If the User Clip Plane0 is enabled,This field contains distance from the vertex to the User Clip Plane1 Format: FLOAT32
D14	31:0	**User Clip Distance to Plane2.** If the User Clip Plane0 is enabled,This field contains distance from the vertex to the User Clip Plane2 Format: FLOAT32
D15	31:0	**User Clip Distance to Plane3.** If the User Clip Plane0 is enabled,This field contains distance from the vertex to the User Clip Plane3 Format: FLOAT32
D16	31:0	**User Clip Distance to Plane4.** If the User Clip Plane0 is enabled,This field contains distance from the vertex to the User Clip Plane4 Format: FLOAT32
D17	31:0	**User Clip Distance to Plane5.** If the User Clip Plane0 is enabled,This field contains distance from the vertex to the User Clip Plane5 Format: FLOAT32

DWord	Bit	Description
D18	31:0	**User Clip Distance to Plane6.** If the User Clip Plane0 is enabled,This field contains distance from the vertex to the User Clip Plane6 Format: FLOAT32
D19	31:0	**User Clip Distance to Plane7.** If the User Clip Plane0 is enabled,This field contains distance from the vertex to the User Clip Plane7 Format: FLOAT32
(D20-Dn)	31:0	(Remainder of Vertex Elements). While DWords D0-D19 are exposed to the device (i.e., read back by FF units), DWords D20-Dn of vertices written (by threads) are opaque to the device. Software is free to format/use these DWords as desired. The absolute maximum size limit on this data is specified via a maximum limit on the amount of data that can be read from a VUE (including the Vertex Header) (Vertex Entry URB Read Length has a maximum value of 63 256-bit units). Therefore the Remainder of Vertex Elements has an absolute maximum size of 62 256-bit units. Of course the actual allocated size of the VUE can and will limit the amount of data in a VUE.

Table 4 VUE Vertex Header ([DevSNB+])

DWord	Bit	Description
D0	31:0	Reserved: MBZ
D1	31:0	**Render Target Array Index (RTAIndex).** This value is (eventually) used to index into a specific element of an "array" Render Target. It is read back by the GS unit (for all exiting vertices) and the Clip unit (for all clip-generated vertices), subsequently routed into the PS thread payload, and eventually included in the RTWrite DataPort message header for use by the DataPort shared function. Software is responsible for ensuring this field is zero whenever a programmable index value is <u>not</u> required. When a programmable index value is required software must ensure that the correct 11-bit value is written to this field. Specifically, the kernels must perform a reange check of computed index values against [0,2047], and output zero if that range is exceeded. Note that the unmodified "renderTargetArrayIndex" must be maintained in the VUE outside of the Vertex Header. Software can force an RTAIndex of 0 to be used (effectively ignoring the setting of this DWord) by use of the **ForceZeroRTAIndex** bit (3DSTATE_CLIP). Otherwise the read-back value will be used to select an RTArray element, after being clamped to the RTArray surface's [**MinimumArrayElement**, **Depth**] range (SURFACE_STATE). Format: 0-based U32 index value

DWord	Bit	Description
D2	31:0	**Viewport Index.** This value is used to select one of a possible 16 sets of viewport (VP) state parameters in the Clip unit's VertexClipTest function and in the SF unit's ViewportMapping and Scissor functions. The GS unit (even if disabled) will read back this value for all vertices exiting the GS stage and entering the Clip stage. When enabled, the GS unit will range-check the value against [0,**Maximum VPIndex]** (see GS_STATE, CLIP_STATE). After this range-check the values are sent down the pipeline and used in the Clip unit's VertexClipTest function. For vertices passing through the Clip stage, these values will also be sent to the SF unit for use in ViewportMapping and Scissor functions. The Clip unit (if enabled) will read back this value only for vertices generated by CLIP threads. The Clip unit will perform a range clamp similar to the GS unit. Software can force a value of 0 to be used by programming **Maximum VPIndex** to 0. Format: 0-based U32 index value
D3	31:0	**Point Width.** This field specifies the width of POINT objects in screen-space pixels. It is used only for vertices within POINTLIST and POINTLIST_BF primitive topologies, and is ignored for vertices associated with other primitive topologies. This field is read back by both the GS and Clip units. Format: FLOAT32
D4	31:0	**Vertex Position X Coordinate.** This field contains the X component of the vertex's 4D space position. Format: FLOAT32
D5	31:0	**Vertex Position Y Coordinate.** This field contains the Y component of the vertex's 4D space position Format: FLOAT32
D6	31:0	**Vertex Position Z Coordinate.** This field contains the Z component of the vertex's NDC space position Format: FLOAT32
D7	31:0	**Vertex Position W Coordinate.** This field contains the Z component of the vertex's 4D space position Format: FLOAT32

DWord	Bit	Description
D8	31:0	**ClipDistance 0 Value (optional).** If the **UserClipDistance Clip Test Enable Bitmask** bit (3DSTATE_CLIP) is set, this value will be read from the URB in the Clip stage. If the value is found to be less than 0 or a NaN, the vertex's UCF<0> bit will set in the Clip unit's VertexClipTest function. If the **UserClipDistance Clip Test Enable Bitmask** bit is clear, this value will not be read back, and the vertex's UCF<0> bit will be zero by definition. Format: FLOAT32
D9	31:0	ClipDistance 1 Value (optional). See above
D10	31:0	ClipDistance 2 Value (optional). See above
D11	31:0	ClipDistance 3 Value (optional). See above
D12	31:0	ClipDistance 4 Value (optional). See above
D13	31:0	ClipDistance 5 Value (optional). See above
D14	31:0	ClipDistance 6 Value (optional). See above
D15	31:0	ClipDistance 7 Value (optional). See above
	31:0	(Remainder of Vertex Elements). The absolute maximum size limit on this data is specified via a maximum limit on the amount of data that can be read from a VUE (including the Vertex Header) (**Vertex Entry URB Read Length** has a maximum value of 63 256-bit units). Therefore the Remainder of Vertex Elements has an absolute maximum size of 62 256-bit units. Of course the actual allocated size of the VUE can and will limit the amount of data in a VUE.

1.5.2 Vertex Positions

(For the sake of brevity, the following discussion will use the term *map* as a shorthand for "compute screen space coordinate via perspective divide followed by viewport transform".)

The "Position" fields of the Vertex Header are the only vertex position coordinates exposed to the 3D Pipeline. The CLIP and SF units are the only FF units which perform operations using these positions. The VUE will likely contain other position attributes for the vertex outside of the Vertex Header, though this information is not directly exposed to the FF units. For example, the Clip Space position will likely be required in the VUE (outside of the Vertex Header) in order to perform correct and robust 3D Clipping in the CLIP thread.

In the CLIP unit, the read-back Position fields are interpreted as being in one of two coordinate systems, depending on the **CLIP_STATE.VertexPositionSpace** bit. The CLIP unit will modify its VertexClipTest function depending on the coordinate space of the incoming vertices.

- **[DevSNB+]: VPOS_CLIPSPACE (Homogeneous 4D Clip-space coordinates, pre-perspective division)**: The Clip Space position is defined in a homogeneous 4D coordinate

space (pre-perspective divide), where the visible "view volume" is defined by the APIs. The API's VS or GS shader program will include geometric transforms in the computation of this clip space position such that the resulting coordinate is positioned properly in relation to the view volume (i.e., it will include a "view transform" in this computation path). When this coordinate system is selected, the 3D FF pipeline will perform a perspective projection (division of x,y,z by w), perform clip-test on the resulting NDC (Normalized Device Coordinates), and eventually perform viewport mapping (in the SF unit) to yield screen-space (pixel) coordinates.

- **VPOS_SCREENSPACE (Screen Space position)**: Under certain circumstances, the position in the Vertex Header will contain the screen-space (pixel) coordinates (post viewport mapping).

The SF unit does not have a state bit defining the coordinate space of the incoming vertex positions. Software must use the Viewport Mapping function of the SF unit in order to ensure that screen-space coordinates are available after that function. If screen space coordinates are passed into SF, then software will likely turn off the Viewport Mapping function.

The following subsections briefly describe the three relevant coordinate spaces.

1.5.2.1 Clip Space Position

The *clip-space* position of a vertex is defined in a homogeneous 4D coordinate space where, after perspective projection (division by W), the visible "view volume" is some canonical (3D) cuboid. Typically the X/Y extents of this cuboid are [-1,+1], while the Z extents are either [-1,+1] or [0,+1]. The API's VS or GS shader program will include geometric transforms in the computation of this clip space position such that the resulting coordinate is positioned properly in relation to the view volume (i.e., it will include a "view transform" in this computation path).

Note that, under typical perspective projections, the clip-space W coordinate is equal to the view-space Z coordinate.

A vertex's clip-space coordinates must be maintained in the VUE up to 3D clipping, as this clipping is performed in clip space.

- In [**DevSNB+**], vertex clip-space positions must be included in the Vertex Header, so that they can be read-back (prior to Clipping) and then subjected to perspective projection (in hardware) and subsequent use by the FF pipeline.

1.5.2.2 NDC Space Position

A perspective divide operation performed on a clip-space position yields a [X,Y,Z,RHW] NDC (Normalized Device Coordinates) space position. Here "normalized" means that visible geometry is located within the [-1,+1] or [0,+1] extent view volume cuboid (see clip-space above).

- The NDC X,Y,Z coordinates are the clip-space X,Y,Z coordinates (respectively) divided by the clip-space W coordinate (or, more correclty, the clip-space X,Y,Z coordinates are multiplied by the reciprocal of the clip space W coordinate).
    - Note that the X,Y,Z coordinates may contain INFINITY or NaN values (see below).

- The NDC RHW coordinate is the reciprocal of the clip-space W coordinate and therefore, under normal perspective projections, it is the reciprocal of the view-space Z coordinate. Note that NDC space is really a 3D coordinate space, where this RHW coordinate is retained in order to perform perspective-correct interpolation, etal. Note that, under typical perspective projections.
    - Note that the RHW coordinate make contain an INFINITY or NaN value (see below).

1.5.2.3 Screen-Space Position

Screen-space coordinates are defined as:

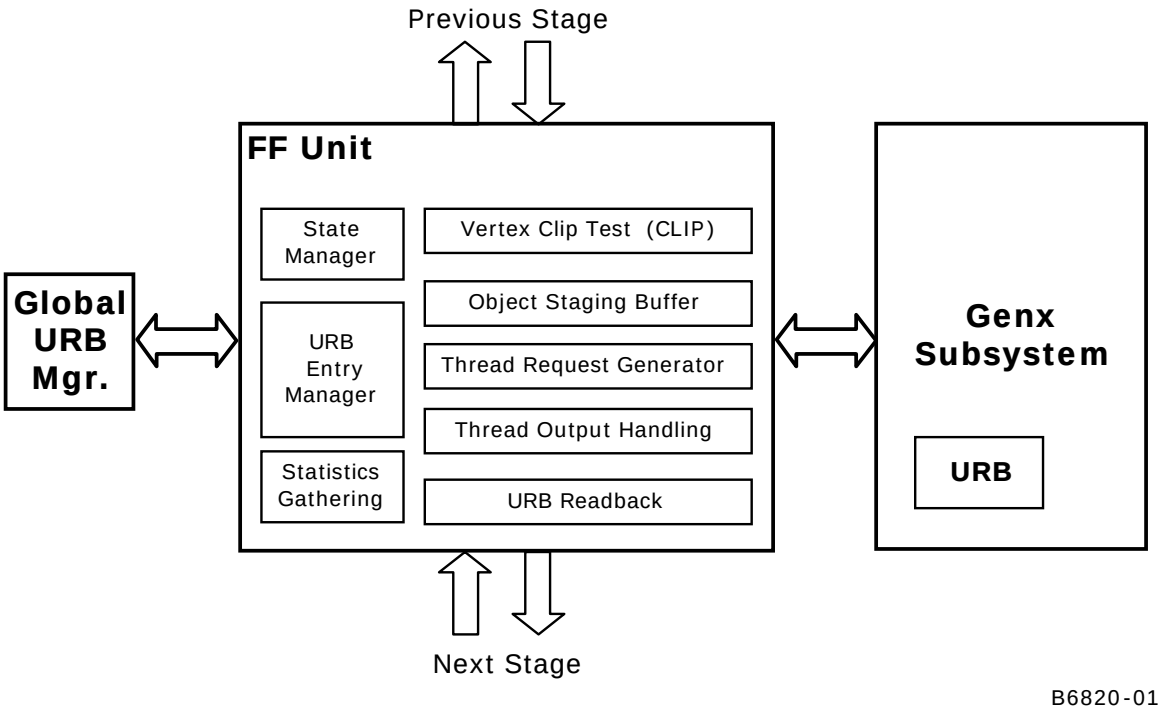
- X,Y coordinates are in absolute screen space (pixel coordinates, upper left origin). See Vertex X,Y Clamping and Quantization in the SF section for a discussion of the limitations/restrictions placed on screenspace X,Y coordinates.
- Z coordinate has been mapped into the range used for DepthTest.
    - D3D allows the visible Z range ([0,1] NDC) to be mapped into some subrange within [0,1]. However, by definition, pre-mapping in D3D disables Z clipping. (If mapped Z coordinates outside of [0,1] are presented, rendering results are undefined.) Software must explicitly disable Z clipping via **Viewport Z ClipTest Enable** (CLIP_STATE) whenever positions are pre-mapped.
- RHW coordinate is actually the reciprocal of clip-space W coordinate (typically the reciprocal of the view-space Z coordinate). D3D requires RHW to be positive, or rendering results are undefined.

1.6 3D Pipeline Stage Overview

The fixed-function (FF) stages of the 3D pipeline share some common functionality, specifically related to the creation and management of threads. This chapter is intended to describe the behavior and programming model of these common functions, in an effort to not replicate this information for each pipeline stage. Stage-specific exceptions to the information provided here will be included in the stage-specific chapters to follow.

1.6.1 Generic 3D FF Unit Block Diagram

The following block diagram, in general, applies to the VS, GS and CLIP stages.



1.6.2 Common 3D FF Unit Functions

A major role of the FF stages is in managing the threads that perform the majority of the processing on the vertex/pixel data. (In general, the amount of non-thread processing performed by the 3DPIPE stages increases towards the end of the pipeline.) In a generic sense, the key functions included are:

- Bypass Mode
- URB Entry Management
- Thread Initiation Management
- Thread Request Data Generation
    - Thread Control Information Generation
    - Thread Payload Header Generation
    - Thread Payload Data Generation
- Thread Output Handling
- URB Entry Readback
- Statistics Gathering

[DevSNB]: The Clip and SF FF units do not dispatch threads.

The following table lists the various state variables used to control the common FF functions:

State Variable	Programmed Via	Generic Functions Affected
<stage> Enable	[DevSNB]: FF inline state	Bypass Mode
Kernel Start Pointer	[DevSNB]: FF inline state	Thread Request Data Gen.
GRF Register Block Count		Thread Request Data Gen.
Single Program Flow		Thread Request Data Gen.
Thread Priority		Thread Request Data Gen.
Floating Point Mode		Thread Request Data Gen.
Exceptions Enable		Thread Request Data Gen.
Scratch Space Base Pointer		Thread Request Data Gen.
Per Thread Scratch Space		Thread Request Data Gen.
Constant URB Entry Read Length		Payload Data Gen.
Constant URB Entry Read Offset		Payload Data Gen.
Vertex URB Entry Read Length		Payload Data Gen.
Vertex URB Entry Read Offset		Payload Data Gen.
Dispatch GRF Start Register for URB Data		Payload Data Gen.
Maximum Number of Threads		Thread Resource Alloc. Scratch Space Mgt.
<stage> Fence	URB_FENCE_POINTER	URB Entry Mgt.
URB Entry Allocation Size	[DevSNB]: 3DSTATE_URB	URB Entry Mgt.
Number of URB Entries		URB Entry Mgt.
Sampler State Pointer	[**DevSNB+**]: 3DSTATE_SAMPLER_ STATE_POINTERS	Payload Header Gen.
<stage> Binding Table Pointer	3DSTATE_BINDING_ TABLE_POINTERS	This gets routed directly to shared functions (transparent to software).

State Variable	Programmed Via	Generic Functions Affected
Sampler Count	[DevSNB]: FF inline state	Thread Request Data Gen.
Binding Table Entry Count		Thread Request Data Gen.
Statistics Enable		Statistics Gathering

1.6.3 Thread Initiation Management

Those FF stages that can spawn threads must have buffered the input (URB entries) available to supply a thread, and then ensure that there are sufficient resources (within the domain of the 3D pipeline) to make the thread request.

Once a FF stage determines a thread request can be submitted, (a) all input data required to initiate the thread is generated, (b) this information is submitted to the common thread dispatcher, (c) the thread dispatcher will spawn the thread as soon as an EU with sufficient GRF resources becomes available, and finally (d) the thread will start execution. With respect to concurrent threads, steps (c) and (d) can proceed out of order (i.e., a threads are not necessarily dispatched in the order that the thread requests are submitted to the thread dispatcher).

1.6.3.1 Thread Input Buffering

Each FF stage varies with regard to thread input requirements, and so this will not be discussed in this chapter other than the overview information provided in the following table:

FF Stage	Thread Input Requirements
CS	N/A (does not spawn threads)
VF	N/A (does not spawn threads)
VS	Normally, two vertices are buffered before a VS thread is spawned to shade the pair in parallel. Under some circumstances (e.g., a flush, state change, etc.) a single vertex will be shaded.
GS	All the vertices associated with an object must be buffered before a GS thread can be initiated to process the object.
CLIP	[DevSNB]: Does not spawn threads.
SF	[DevSNB]: Does not spawn threads.
WM	Threads spawned as required by the rasterization algorithm.

1.6.3.2 Thread Resource Allocation [Pre-DevIL]

Once a FF stage that spawn threads has sufficient input to initiate a thread, it must guarantee that it is safe to request the thread initiation. For all these FF stages, this check is based on :

- **The availability of output URB entries**:

  - VS: As the input URB entries are overwritten with the VS-generated output data, output URB availability isn't a factor.

  - GS: At least one output URB entry must be available to serve as the initial output vertex from the GS thread. <u>However</u>, software must <u>guarantee</u> that additional URB entries will <u>eventually become available</u> to allow the pipeline to make forward progress and not deadlock. There are two considerations here:

    - Single GS Threads (**Maximum Number of Threads** == 1): There must be enough GS output URB entries allocated to allow the GS thread to make progress (call this number P). P must include enough vertices to allow the next enabled stage to make progress, i.e., must contain enough vertices for the worst-case object within a primitive. For example, the system would hang if the GS stage was only allocated 2 URB entries and the GS thread tried to output a TRILIST. In this case the GS stage would need to be allocated at least 3 URB entries – the GS thread would output the first 3 vertices, then would stall on the allocation of the 4th vertex until the rest of the pipeline consumed that first triangle and dereferenced the first vertex. The clipper, when enabled, imposes additional requirements on the number of output URB entries allocated to the GS. Because of the way the clipper processes strip/fan primitives, it will not release the URB entries for the vertices of a given object until it has finished processing the *next* object in the primitive. The minimum number of handles that must be allocated to the GS for strip/fan –type primitives is thus increased according to the following table:

Topology	Minimum GS Handles
LINESTRIP, LINESTRIP_BF, LINESTRIP_CONT, LINESTRIP_CONT_BF	3
POLYGON, TRIFAN, TRIFAN_NOSTIPPLE	4
TRISTRIP, TRISTRIP_REV	5

        - Dual GS threads: If two concurrent GS thread are permitted, software must account for the possibility that the subsequent GS thread completes before the preceding GS thread outputs its first vertex. Therefore there must be enough URB entries allocated to satisfy the above minimums for both threads.

    - CLIP: Same considerations as GS (above)
    - SF: An output URB entry must be available to store the results of the SETUP thread.
    - WM: N/A (does not output to URB)

- **The Maximum Number of Threads** state variable. This state variable limits the number of concurrent threads a FF stage can have executing. As long as the FF stage is operating below this limit, it can make additional thread initiation requests.
- In addition, the WM unit utilizes a **scoreboard** mechanism to ensure proper ordering of operations – and this mechanism can postpone the initiation of new threads. (See Windower chapter).

Software is responsible for programming of **Maximum Number of Threads** to ensure the correct and optimal operation of the 3D pipeline.

The considerations for programming **Maximum Number of Threads** are summarized below:

1. **URB Allocation**: (See discussion above)
2. **Scratch Space Allocation**: When the current kernel of an enable stage requires use of scratch space (for API-defined temporary storage, register spill/fill, overflow stacks, etc.), software must limit the number of concurrent threads (via **Maximum Number of Threads**) such that the total scratch space requirement is satisfied by the amount of scratch space memory allocated to the FF stage.
3. **Stream Output Serialization**: If a kernel is required to output a serialized stream of data to a memory buffer, threads for that stage must be serialized by SW only allowing (**Maximum Number of Threads** == 1).
4. **Performance**: In general, a larger number of possibly-concurrent threads will better ensure the cores are fully utilized.

(***Note***: The 3D pipeline can function correctly with (**Maximum Number of Threads** == 1) set at each enabled stage, given that there are sufficient resources to run this single thread (scratch space, etc). However, this will certainly not be an optimal configuration. See *Graphics Processing Engine* for a discussion of URB Allocation Requirements and Guidelines which includes information on programming the Number Of Threads for the various FF units.)

1.6.3.3 Thread Resource Allocation [DevSNB+]

In general, the considerations listed in the preceding DevIL section are relevant, with the following exception: CLIP, SF: Threads are not spawned.

1.6.4 Thread Request Generation

Once a FF unit determines that a thread can be requested, it must gather all the information required to submit the thread request to the Thread Dispatcher. This information is divided into several categories, as listed below and subsequently described in detail.

- **Thread Control Information**: This is the information required (from the FF unit) to establish the execution environment of the thread. Note that some information affecting the thread execution state is programmed external to the 3D pipeline (e.g., Exception Handler IP, Breakpoint IP, etc.)

- **Thread Payload Header**: This is the first portion of the thread payload passed in the GRF, starting at GRF R0. This is information passed directly from the FF unit. It precedes the Thread Payload Input URB Data.

- **Thread Payload Input URB Data**: This is the second portion of the thread payload. It is read from the URB using entry handles supplied by the FF unit.

1.6.4.1 Thread Control Information

The following table describes the various state variables that a FF unit uses to provide information to the Thread Dispatcher and which affect the thread execution environment. Note that this information is not directly passed to the thread in the thread payload (though some fields may be subsequently accessed by the thread via architectural registers).

Table 5. State Variables Included in Thread Control Information

State Variable	Usage	FFs
Kernel Start Pointer	This field, together with the **General State Pointer**, specifies the starting location (1st core instruction) of the kernel program run by threads spawned by this FF unit. It is specified as a 64-byte-granular offset from the **General State Pointer**.	All FFs spawning threads

State Variable	Usage	FFs
GRF Register Block Count	Specifies, in 16-register blocks, how many GRF registers are required to run the kernel. The Thread Dispatcher will only seek candidate EUs that have a sufficient number of GRF register blocks available. Upon selecting a target EU, the Thread DIspatcher will generate a logical-to-physical GRF mapping and provide this to the target EU.	All FFs spawning threads
Single Program Flow (SPF)	Specifies whether the kernel program has a single program flow (SIMDnxm with m = 1) or multiple program flows (SIMDnxm with m > 1). See CR0 description in *ISA Execution Environment*.	All FFs spawning threads
Thread Priority	The Thread Dispatcher will give priority to those thread requests with Thread Priority of HIGH_PRIORITY over those marked as LOW_PRIORITY. Within these two classes of thread requests, the Thread Dispatcher applies a priority order (e.g., round-robin --- though this algorithm is considered a device implementation-dependent detail).	All FFs spawning threads
Floating Point Mode	This determines the initial value of the **Floating Point Mode** bit of the EU's CR0 architectural register that controls floating point behavior in the EU core. (See ISA.)	All FFs spawning threads
Exceptions Enable	This bitmask controls the exception hanlding logic in the EU. (See ISA.)	All FFs spawning threads
Sampler Count	This is a hint which specifies how many indirect SAMPLER_STATE structures should be prefetched concurrent with thread initiation. It is recommended that software program this field to equal the number of samplers, though there may be some minor performance impact if this number gets large. This value should not exceed the number of samplers accessed by the thread as there would be no performance advantage. Note that the data prefetch is treated as any other memory fetch (with respect to page faults, etc.).	All stages supporting sampling (VS, GS, WM)
Binding Table Entry Count	This is a hint which specifies how many indirect BINDING_TABLE_STATE structures should be prefetched concurrent with thread initiation. (The comments included in Sampler Count (above) also apply to this field).	All FFs spawning threads

1.6.4.2 Thread Payload Generation

FF units are responsible for generating a thread *payload* – the data pre-loaded into the target EU's GRF registers (starting at R0) that serves as the primary direct input to a thread's kernel. The general format of these payloads follow a similar structure, though the exact payload size/content/layout is unique to each stage. This subsection describes the common aspects – refer to the specific stage's chapters for details on any differences.

The payload data is divided into two main sections: the *payload header* followed by the *payload URB data*. The payload header contains information passed directly from the FF unit, while the payload URB data is obtained from URB locations specified by the FF unit.

NOTE: The first 256 bits of the thread payload (the initial contents of R0, aka "the R0 header") is specially formatted to closely match (and in some cases exactly match) the first 256 bits of thread-generated *messages* (i.e., the message header) accepted by shared functions. In fact, the send instruction supports having a copy of a GR's contents (such as R0) used as the message header. Software must take this intention into account (i.e., "don't muck with R0 unless you know what you're doing"). This is especially important given the fact that several fields in the R0 header are considered opaque to SW, where use or modification of their contents might lead to UNDEFINED results.

The payload header is further (loosely) divided into a leading *fixed payload header* section and a trailing, variable-sized *extended payload header* section. In general the size, content and layout of both payload header sections are FF-specific, though many of the fixed payload header fields are common amongst the FF stages. The extended header is used by the FF unit to pass additional information specific to that FF unit. The extended header is defined to start after the fixed payload header and end at the offset defined by **Dispatch GRF Start Register for URB Data**. Software can cause use the **Dispatch GRF Start Register for URB Data** field to insert padding into the extended header in order to maintain a fixed offset for the start of the URB data.

1.6.4.2.1 Fixed Payload Header

The payload header is used to pass FF pipeline information required as thread input data. This information is a mixture of SW-provided state information (state table pointers, etc.), primitive information received by the FF unit from the FF pipeline, and parameters generated/computed by the FF unit. most of the fields of the fixed header are common between the FF stages. These non-FF-specific fields are described in Table 6. Note that a particular stage's header may not contain all these fields, so they are not "common" in the strictest sense.

Table 6. Fixed Payload Header Fields (non-FF-specific)

Fixed Payload Header Field (non-FF-specific)	Description	FFs
FF Unit ID	Function ID of the FF unit. This value identifies the FF unit within the subsystem. The FF unit will use this field (when transmitted in a Message Header to the URB Function) to detect messages emanating from its spawned threads.	All FFs spawning threads
Reserved	--	

Fixed Payload Header Field (non-FF-specific)	Description	FFs
Thread ID	This field uniquely identifies this thread within the FF unit over some period of time.	All FFs spawning threads
Scratch Space Pointer	This is the starting location of the thread's allocated scratch space, specified as an offset from the **General State Base Address**. Note that scratch space is allocated by the FF unit on a per-thread basis, based on the **Scratch Space Base Pointer** and **Per-Thread Scratch Space Size** state variables. FF units will assign a thread an arbitrarily-positioned region within this space. The scratch space for multiple (API-visible) entities (vertices, pixels) will be interleaved within the thread's scratch space.	All FFs spawning threads
Dispatch ID	This field identifies this thread within the outstanding threads spawned by the FF unit. This field does not uniquely identify the thread over any significant period of time. *Implementation Note*: This field is effectively an "active thread index". It is used on a thread's URB allocation request to identify which thread's handle pool is to source the allocation. It is used upon thread termination to free up the thread's scratch space allocation.	All FFs spawning threads
Binding Table Pointer	This field, together with the **Surface State Base Pointer**, specifies the starting location of the Binding Table used by threads spawned by the FF unit. It is specified as a 64-byte-granular offset from the **Surface State Base Pointer**. See *Shared Functions* for a description of a Binding Table.	All FFs spawning threads
Sampler State Pointer	This field, together with the **General State Base Pointer**, specifies the starting location of the Sampler State Table used by threads spawned by the FF unit. It is specified as a 64-byte-granular offset from the **General State Base Pointer**. See *Shared Functions* for a description of a Sampler State Table.	All FFs spawning threads which sample (VS, GS, WM)
Per Thread Scratch Space	This field specifies the amount of scratch space allocated to each thread spawned by the FF unit. The driver must allocate enough contiguous scratch space, starting at the **Scratch Space Base Pointer**, to ensure that the **Maximum Number of Threads** can each get **Per-Thread Scratch Space** size without exceeding the driver-allocated scratch space.	All FFs spawning threads

Fixed Payload Header Field (non-FF-specific)	Description	FFs
Handle ID <n>	This ID is assigned by the FF unit and links the thread to a specific entry within the FF unit. The FF unit will use this information upon detecting a URB_WRITE message issued by the thread. Threads spawned by the GS, CLIP, and SF units are provided with a single Handle ID / URB Return Handle pair. Threads spawned by the VS are provided with one or two pairs (depending on how many vertices are to be processed). Threads spawned by the WM do not write to URB entries, and therefore this info is not supplied.	VS,GS,CLIP,SF
URB Return Handle <n>	This is an initial destination URB handle passed to the thread. If the thread does output URB entries, this identifies the destination URB entry. hreads spawned by the GS, CLIP, and SF units are provided with a single Handle ID / URB Return Handle pair. Threads spawned by the VS are provided with one or two pairs (depending on how many vertices are to be processed). Threads spawned by the WM do not write to URB entries, and therefore this info is not supplied.	VS,GS,CLIP,SF
Primitive Topology Type	As part of processing an incoming primitive, a FF unit is often required to spawn a number of threads (e.g., for each individual triangle in a TRIANGLE_STRIP). This field identifies the type of primitive which is being processed by the FF unit, and which has lead to the spawning of the thread. Kernels written to process different types of objects can use this value to direct that processing. E.g., when a CLIP kernel is to provide clipping for all the various primitive types, the kernel would need to examine the Primitive Topology Type to distinguish between point, lines, and triangle clipping requests. **NOTE**: In general, this field is identical to the Primitive Topology Type assoociated with the primitive vertices as received by the FF unit. Refer to the individual FF unit chapters for cases where the FF unit modifies the value before passing it to the thread. (E.g., certain units perform toggling of TRIANGLESTRIP and TRIANGLESTRIP_REV).	GS, CLIP, SF, WM

1.6.4.2.2 Extended Payload Header

The extended header is of variable-size, where inclusion of a field is determined by FF unit state programming.

In order to permit the use of common kernels (thus reducing the number of kernels required), the **Dispatch GRF Start Register for URB Data** state variable is supported in all FF stages. This SV is used to place the payload URB data at a specific starting GRF register, irrespective of the size of the extended header. A kernel can therefore reference the payload URB data at fixed GRF locations, while conditionally referencing extended payload header information.

1.6.4.2.3 Payload URB Data

In each thread payload, following the payload header, is some amount of URB-sourced data required as input to the thread. This data is divided into an optional *Constant URB Entry* (CURBE), following either by a Primitive URB Entry (WM) or a number of Vertex URB Entries (VS, GS, CLIP, SF). A FF unit only knows the location of this data in the URB, and is never exposed to the contents. For each URB entry, the FF unit will supply a sequence of handles, read offsets and read lengths to the subsystem. The subsystem will read the appropriate 256-bit locations of the URB, optionally perform swizzling (VS only), and write the results into sequential GRF registers (starting at **Dispatch GRF Start Register for URB Data**).

Table 7. State Variables Controlling Payload URB Data

State Variable	Usage	FFs
Dispatch GRF Start Register for URB Data	This SV identifies the starting GRF register receiving payload URB data. Software is responsible for ensuring that URB data does not overwrite the Fixed or Extended Header portions of the payload.	FFs spawning threads
Vertex URB Entry Read Offset	This SV specifies the starting offset within VUEs from which vertex data is to be read and supplied in this stage's payloads. It is specified as a 256-bit offset into any and all VUEs passed in the payload. This SV can be used to skip over leading data in VUEs that is not required by the stage's threads (e.g., skipping over the Vertex Header data at the SF stage, as that information is not required for setup calculations). Skipping over irrelevant data can only help to improve performance. Specifying a vertex data source extending beyond the end of a vertex entry is UNDEFINED.	VS, GS,
Vertex URB Entry Read Length	This SV determines the amount of vertex data (starting at **Vertex URB Entry Read Offset**) to be read from each VUEs and passed into the payload URB data. It is specified in 256-bit units. A zero value is INVALID (at very least one 256-bit unit must be read). Specifying a vertex data source extending beyond the end of a VUE is UNDEFINED.	

Programming Restrictions: (others may already have been mentioned)

- The maximum size payload for any thread is limited by the number of GRF registers available to the thread, as determined by min(128, 16 * **GRF Register Block Count**). Software is responsible for ensuring this maximum size is not exceeded, taking into account:
    - The size of the Fixed and Extended Payload Header associated with the FF unit.
    - The **Dispatch GRF Start Register for URB** Data SV.
    - The amount of CURBE data included (via **Constant URB Entry Read Length**)
    - The number of VUEs included (as a function of FF unit, it's state programming, and incoming primitive types)
    - The amount of VUE data included for each vertex (via **Vertex URB Entry Read Length**)
    - (For WM-spawned PS threads) The amount of Primitive URB Entry data.
- For any type of URB Entry reads:

- Specifying a source region (via Read Offset, Read Length) that goes past the end of the URB Entry allocation is illegal.
    - The allocated size of Vertex/Primitive URB Entries is determined by the **URB Entry Allocation Size** value provided in the pipeline state descriptor of the FF unit owning the VUE/PUE.
    - The allocated size of CURBE entries is determined by the **URB Entry Allocation Size** value provided in the CS_URB_STATE command.

1.6.5 Thread Output Handling

Those FF units spawning threads are responsible for monitoring and responding to certain events generated by their spawned threads. Such events are indirectly detected by these FF units monitoring messages sent from threads to the URB Shared Function. By snooping the Message Bus Sideband and Header information, a FF can detect when a particular spawned thread sends a message to the URB function. A subset of this information is then captured and acted upon. Refer to the *URB* chapter for more details (including a table of valid/invalid combinations of the **Complete**, **Used**, **Allocate**, and **EOT** bits)

The following subsections describe functions that FF units perform as part of Thread Output Handling.

1.6.5.1 URB Entry Output (VS, GS)

The following description is applicable only to the VS and GS stages.

For these threads the main (if not only) output of the thread takes the form of data written to one or more destination VUEs. At very least this is the only form of thread output visible to the FF units.

When a thread sends a URB_WRITE message to the URB function with the **Complete** and **Used** bits set in the Message Description, the spawning FF unit recognizes this as the thread having completely written a destination UE. (In the typical case of a VS thread, a pair of UEs will be written in parallel). The thread must not target any additional URB messages to this UE (unless it gets reallocated to the thread). The FF unit marks this UE as complete and available for output.

In the case where multiple concurrent threads are supported at a given stage, the FF unit is responsible for outputing UEs down the pipeline in order. I.e., all VUE outputs of a spawned thread must be sent down the pipeline (in order of allocation to the thread) prior to any outputs from a subsequently-spawned thread. This is required even if the subsequent threads perform any/all of their output prior to the preceding thread producing any/some output.

1.6.5.2 VUE Allocation (GS, CLIP) [Pre-DevIL]

The following description is applicable only to the GS, CLIP stages.

The GS and CLIP threads are passed a single, initial destination VUE handle. These threads may be required to output more than one destination VUE, and therefore they are provided with a mechanism to request additional handles.

When a GS or CLIP thread issues a URB_WRITE message with the **Allocate** bit set, the spawning FF unit will consider this a request for the allocation of an additional VUE handle. The thread must specify a destination GRF register for the message writeback data. The spawning FF unit will perform the

allocation, and provide the writeback data (containing **Handle ID** and **URB Return Handle**) to the subsystem, which will in turn deliver that data to the appropriate GRF register. (See the *URB* chapter for the definition of this writeback data).

The thread is allowed to proceed while the allocation is taking place (it is guaranteed to complete at some point). If the thread attempts to reference the writeback data before the allocation has completed, execution will be stalled in the same fashion any unfulfilled dependency is handled. It is therefore recommended that SW (a) request the additional allocation as soon as possible, and (b) reference the writeback data as late as possible in order to keep the thread in a runnable state. (Refer to the following subsection to see how the thread is allowed to "allocate ahead" and give back unused VUE handles).

NOTE: GS and CLIP threads must write VUEs in the order they are allocated by the FF unit (in response to an allocation request from the thread), starting with the initial destination handle passed in the thread payload.

A GS or CLIP thread is restricted as to the number of URB handles it can retain. Here a "retained" handle refers to a URB handle that (a) has been pre-allocated or allocated and returned to the thread via the **Allocate** bit in the URB_WRITE message, and (b) has yet to be returned to the pipeline via the **Complete** bit in the URB_WRITE message.

- When operating in single-thread mode (**Maximum Number of Threads** == 1), the number of retained handles must not exceed min(16, **Number of URB Entries**).
- When operating in dual-thread mode (**Maximum Number of Threads** == 2), the number of retained handles must not exceed (**Number of URB Entries**/2).

This restriction is not expected to be significant in that most/all GS/CLIP threads are expected to retain only a few (<=4) handles.

1.6.5.3 VUE Allocation (GS, CLIP) [DevIL]

The following description is applicable only to the GS, CLIP stages.

The threads are not passed an initial handle. Instead, they request a first handle (if any) via the URB shared function's FF_SYNC message (see Shared Functions). If additional handles are required, the URB_WRITE allocate mechanism (mentioned above) is used.

1.6.5.4 VUE Allocation (GS) [DevSNB+]

The following description is applicable only to the GS stage.

The threads are not passed an initial handle. Instead, they request a first handle (if any) via the URB shared function's FF_SYNC message (see Shared Functions). If additional handles are required, the URB_WRITE allocate mechanism (mentioned above) is used.

1.6.5.5 VUE Dereference (GS)

The following description is applicable only to the GS stage.

It is possible and legal for a thread to produce no output or subsequently allocate a destination VUE that was not required (e.g., the thread allocated ahead). Therefore, there is a mechanism by which a thread

can “give back” (dereference) an allocated VUE. This mechanism must be used if the VUE is not written before the thread terminates.

A kernel can explicitly dereference a VUE by issuing a URB_WRITE message (specifying the to-be-dereference handle) with the **Complete** bit set and the **Used** bit clear.

1.6.5.6 Thread Termination

All threads must explicitly terminate by executing a SEND instruction with the EOT bit set. (See *EU* chapters). When a thread spawned by a 3D FF unit terminates, the spawning FF unit detects this termination as a part of Thread Management. This allows the FF units to manage the number of concurrent threads it has spawned and also manage the resources (e.g., scratch space) allocated to those threads.

Programming Note: [Pre-DevIL] GS and Clip threads must terminate by sending a URB_WRITE message (with EOT set) with the Complete bit also set (therein returning a URB handle marked as either used or un-used).

1.6.6 VUE Readback

Starting with the CLIP stage, the 3D pipeline requires vertex information in addition to the VUE handle. For example, the CLIP unit’s VertexClipTest function needs the vertex position, as does the SF unit’s functions. This information is obtained by the 3D pipeline reading a portion of each vertex’s VUE data directly from the URB. This readback (effectively) occurs immediately before the CLIP VertexClipTest function, and immediately after a CLIP thread completes the output of a destination VUE.

The Vertex Header (first 256 bits) of the VUE data is read back. (See the previous *VUE Formats* subsection (above) for details on the content and format of the Vertex Header.) [**DevSNB+**]: Additional Clip/Cull data (located immediately past the Vertex Header) may be read prior to clipping.

This readback occurs automatically and is not under software control. The only software implication is that the Vertex Header must be valid at the readback points, and therefore must have been previously loaded or written by a thread.

1.6.7 Statistics Gathering [DevSNB]

The Vertex Fetch, Geometry Shader and Clipper units count the number of complete primitives that they issue down the pipeline. The Vertex Fetch unit counts the number of vertices and objects it issues. The Vertex Shader, Geometry Shader, [Pre-DevIL] Clipper and Windower keep a count of the number of objects they pass to shader threads. The Windower counts the number of pixels that turn out to be visible after stencil and depth testing (the Color Calculator also helps track this statistic.)

The pipeline must be completely flushed prior to reading out the values of these counters via MMIO (or MI_STORE_REGISTER_MEM) and reporting them to the API. Without a flush it is impossible to tell which work in the pipeline has affected a given statistic, and which has not.

These statistics counters are initialized by writing the value 0 to them via MMIO or MI_LOAD_REGISTER_IMM. Generally this should be done only at API “pipeline creation”. Each context has its own statistics so these registers are saved and restored on context switch. Table 8 shows the

statistics counter register names and MMIO offsets. See the *Memory Interface Registers* chapter for more detailed register information.

Table 8. Statistics MMIO Registers

MMIO Register	Statistic	Controlled By
IA_PRIMITIVES_COUNT	VF Primitives Output	VF
IA_VERTICES_COUNT	VF Vertices Output	VF
VS_INVOCATION_COUNT	VS Vertices Shaded	VS
GS_INVOCATION_COUNT	Geometry GS Threads	GS
GS_PRIMITIVES_COUNT	[Pre-DevIL]: Geometry Primitives Output [DevIL+]: Accumulation of GS Shader-supplied GS_PRIMITIVES count	[Pre-DevIL]: Clip [DevIL+]: GS thread via URB_WRITE
[DevCTG+]: SO_NUM_PRIMS_WRITTEN	Stream Output Primitives Written	[DevCTG]: GS thread via SVBWrite [DevIL+]: GS thread via URBWrite
[DevCTG+]: SO_PRIM_STORAGE_NEEDED	Stream Output Primitives Storage Needed	[DevCTG]: GS thread via SVBWrite [DevIL+]: GS thread via FF_SYNC
CL_INVOCATION_COUNT	[Pre-DevCTG]: Clipper Clip Threads [DevCTG]: Under GS kernel control. See URB_WRITE. [DevIL+]: Clipper Input Primitives	CL
CL_PRIMITIVES_COUNT	Primitives Output from Clip unit to SF unit	SF
PS_INVOCATION_COUNT	Windower Pixels Shaded	WM
PS_DEPTH_COUNT	“Visible” pixels	WM+CC

All the 3D FF units perform some part of the statistics gathering. At the 3D FF unit level, this function is controlled by the **Statistics Enable** bit in each unit’s pipeline state (except VF, which has no pipeline

state and uses a dedicated command). Refer to the individual FF unit chapters for details on the statistics gathered.

Tracking of these statistics should be enabled by SW anytime the 3D pipeline is operating. A control to disable statistics gathering is provided in case the driver wishes to render primitives that are not initiated by the API (to support a stretch blit, for example). Statistics are gathered on behalf of the API and primitives it does not initiate should not affect the statistics in any way. Each FF unit has an individual control to disable statistics gathering. Normally these controls should all be set and reset as a group; in other words the **Statistics Enable** bits in the different FFs state descriptors that are loaded with one PSP command should be the same. The individual controls exist only to make the hardware implementation more straightforward. A single control would require state shared amongst all the FF units, something that isn't currently supported.

[Pre-DevCTG]:

There is a mismatch between what DX10 requires for certain pipeline statistics counters and what the device is counting.

- DX10
    - **GSPrimitives**: DX10 requires a count of primitives output by GS shaders. This does not include primitives flowing thru the GS stage when the GS shader is NULL, at least when StreamOutput is also disabled. (Whether or not streamed-out primitives are counted when the GS shader is NULL is undefined).
    - **CInvocations**: DX10 requires a count of primitives that are submitted for rasterization (which starts with clip-test/clipping), regardless of whether they are trivially-accepted, trivially-rejected, or must-clip cases. Primitives issued when rasterization is disabled are not counted.
- **[Pre-DevCTG]**
    - **GS_PRIMITIVES_COUNT**: The device counts primitives which reach the Clip FF unit, as enabled via SF_STATE.**GSOutputObjectStatisticEnable**.
        - SW can use this counter to support GSPrimitives assuming it is enabled only when the GS shader is enabled.
        - SW can use this counter to support CInvocations assuming it is enabled given the following exceptions/caveats: (a) it is disabled when the driver renders non-app-issued primitives, and (b) when rasterization is disabled either the counter is disabled or software must ensure that primitives are not emitted by the GS unit.
    - **CLIP_INVOCATIONS_COUNT**: This HW counter is counting Clip thread dispatches. There is no corresponding DX10 pipeline statistic counter, so this counter is effectively useless except for internal uses (perfmon, etc.).

Given the above descriptions, software can use GS_PRIMITIVES_COUNT to support either GSPrimitives or CInvocations, but not both simultaneously. (CLIP_INVOCATIONS_COUNT cannot be used to support either.) Therefore software needs to get creative to support "the other" counter.

Given that GSPrimitives should only include GS shader-produced primitives, it seems natural for software to (a) use the GS kernel to support GSPrimitives and (b) use GS_PRIMITIVES_COUNT to support CInvocations (which includes GS-enabled and GS-disabled primitives). The GS kernel could be enhanced to increment a per-context, memory-resident GSPrimitives counter, using DataPort to

read/write the counter from the GS thread. This is similar to how the GS kernel implements the SVBI indices and the StreamOutput statistics counters. In order to permit two concurrent GS threads, the GS kernel can use the **FFTID** bit of the GS thread payload to modify one of two thread-slot-specific counters in memory. Without this (or similar) mechanism, only one outstanding GS thread could be permitted at any given time in order to prevent collisions on incrementing a single memory-resident counter.

1.7 Synchronization of the 3D Pipeline

Two types of synchronizations are supported for the 3D pipe: top of the pipe and end of the pipe. Top of the pipe synchronization really enforces the read-only cache invaliadation. This synchronization guarantees that primitives rendered after such synchronization event fetches the latest read-only data from memory. End of the pipe synchronization enforces that the read and/or read-write buffers do not have outstanding hardware accesses. These are used to implement read and write fences as well as to write out certain statistics deterministically with respect to progress of primitives through the pipeline (and without requiring the pipeline to be flushed.) The PIPE_CONTROL command (see details below) is used to perform all of above synchronizations.

1.7.1 Top-of-Pipe Synchronization

The driver can use top-of-pipe synchronization to invalidate read-only caches in hardware. This operation is performed only after determining that no pending accesses from the hardware exist on these read-only buffers. PIPE-CONTROL aommnd described below allows for invalidating individual read-only buffer type. It is recommended that driver invalidates only the required caches on the need basis so that cache warm-up overhead can be reduced.

1.7.2 End-of-Pipe Synchronization

The driver can use end-of-pipe synchronization to know that rendering is complete (although not necessarily in memory) so that it can de-allocate in-memory rendering state, read-only surfaces, instructions, and constant buffers. An end-of-pipe synchronization point is also sufficient to guarantee that all pending depth tests have completed so that the visible pixel count is complete prior to storing it to memory. End-of-pipe completion is sufficient (although not necessary) to guarantee that read events are complete (a “read fence” completion). Read events are still pending if work in the pipeline requires any type of read except a render target read (blend) to complete.

Write synchronization is a special case of end-of-pipe synchronization that requires that the render cache and/or depth erlated caches are flushed to memory, where the data will become globally visible. This type of synchronization is required prior to SW (CPU) actually reading the result data from memory, or initiating an operation that will use as a read surface (such as a texture surface) a previous render target and/or depth/stencil buffer.

1.7.3 Synchronization Actions

In order for the driver to act based on a synchronization point (usually the whole point), the reaching of the synchronization point must be communicated to the driver. This section describes the actions that may be taken upon completion of a synchronization point which can achieve this communication.

1.7.3.1 Writing a Value to Memory

The most common action to perform upon reaching a synchronization point is to write a value out to memory. An immediate value (included with the synchronization command) may be written. In lieu of an immediate value, the 64-bit value of the PS_DEPTH_COUNT (visible pixel count) or TIMESTAMP register may be written out to memory. The captured value will be the value at the moment all primitives parsed prior to the synchronization commands have been completely rendered, and optionally after all said primitives have been pushed to memory. It is not required that a value be written to memory by the synchronization command.

Visible pixel or TIMESTAMP information is only useful as a delta between 2 values, because these counters are free-running and are not to be reset except at initialization. To obtain the delta, two PIPE_CONTROL commands should be initiated with the command sequence to be measured between them. The resulting pair of values in memory can then be subtracted to obtain a meaningful statistic about the command sequence.

1.7.3.1.1 PS_DEPTH_COUNT

If the selected operation is to write the visible pixel count (PS_DEPTH_COUNT register), the synchronization command should include the **Depth Stall Enable** parameter. There is more than one point at which the global visible pixel count can be affected by the pipeline; once the synchronization command reaches the first point at which the count can be affected, any primitives following it are stalled at that point in the pipeline. This prevents the subsequent primitives from affecting the visible pixel count until all primitives preceding the synchronization point reach the end of the pipeline, the visible pixel count is accurate and the synchronization is completed. This stall has a minor effect on performance and should only be used in order to obtain accurate “visible pixel” counts for a sequence of primitives.

The PS_DEPTH_COUNT count can be used to implement an (API/DDI) “Occlusion Query” function.

1.7.3.2 Generating an Interrupt

The synchronization command may indicate that a “Sync Completion” interrupt is to be generated (if enabled by the MI Interrupt Control Registers – see *Memory Interface Registers*) once the rendering of all prior primitives is complete. Again, the completion of rendering can be considered to be when the internal render cache has been updated, or when the cache contents are visible in memory, as selected by the command options.

1.7.3.3 Invalidating of Caches

If software wishes to use the notification that a synchronization point has been reached in order to reuse referenced structures (surfaces, state, or instructions), it is not sufficient just to make sure rendering is complete. If additional primitives are initiated after new data is laid over the top of old in memory following a synchronization point, it is possible that stale cached data will be referenced for the subsequent rendering operation. In order to avoid this, the PIPE_CONTROL command must be used. (See PIPE_CONTROL description below).

1.7.4 PIPE_CONTROL Command

The PIPE_CONTROL command is used to effect the synchronization described above. Parsing of a PIPE_CONTROL command stalls 3D pipe only if the stall enable bit is set. Commands after

PIPE_CONTROL will continue to be parsed and processed in the 3D pipeline. This may include additional PIPE_CONTROL commands. The implementation does enforce a practical upper limit [DevSNB B+] (8) [DevSNB A] (7) on the number of PIPE_CONTROL commands that may be outstanding at once. Parsing of a PIPE_CONTROL command that causes this limit to be reached will stall the parsing of new commands until the first of the outstanding PIPE_CONTROL commands reaches the end of the pipe and retires.

Note that although PIPE_CONTROL is intended for use with the 3D pipe, it *is* legal to issue PIPE_CONTROL when the Media pipe is selected. In this case PIPE_CONTROL will stall at the top of the pipe until the Media FFs finish processing commands parsed before PIPE_CONTROL. Post-synchronization operations, flushing of caches and interrupts will then occur if enabled via PIPE_CONTROL parameters. Due to this stalling behavior, only one PIPE_CONTROL command can be outstanding at a time on the Media pipe.

[**DevCTG+**]: For the invalidate operation of the pipe control, the following pointers are affected. The invalidate operation affects the restore of these packets. If the pipe control invalidate operation is completed before the context save, the indirect pointers will not be restored from memory.

1. Pipeline State Pointer
2. Media State Pointer
3. Constant Buffer Packet

[DevSNB+] Vertex caches are only invalidated when the VF invalidate bit is set in PIPE_CONTROL (i.e. decision is done in software, not hardware) Note that the index-based vertex cache is always flushed between primitive topologies and of course PIPE_CONTROL can only be issued between primitive topologies. Therefore only the VF ("address-based") cache is uniquely affected by PIPE_CONTROL.

1.7.4.1 PIPE_CONTROL [DevSNB+]

- **[DevSNB B+]** Hardware can support up to 8 pending PIPE_CONTROL flushes
- **[DevSNB A]** Hardware can support 7 pending PIPE_CONTROL flushes
- **[DevSNB:A{W/A}]** When performing a PIPE_CONTROL with TLB invalidate, the driver must follow the current programming below. Without this, hardware cannot guarantee the command after the PIPE_CONTROL w/ TLB inv will not use the old TLB values.

Ring/Batch Contents [DevSNB:A]
PIPE_CONTROL w/ stall (20) and TLB inv bit (18) set
6 Store Data Commands (such as MI_STORE_DATA_IMM or MI_STORE_DATA_INDEX)
PIPE_CONTROL w/ stall bit (20) set

- **[DevSNB:B+{W/A}]** When performing a PIPE_CONTROL with TLB invalidate, the driver must follow the current programming below. Without this, hardware cannot guarantee the command after the PIPE_CONTROL w/ TLB inv will not use the old TLB values.

Ring/Batch Contents [DevSNB:B/C+]
2 Store Data Commands (such as MI_STORE_DATA_IMM or MI_STORE_DATA_INDEX)
PIPE_CONTROL w/ stall (20) and TLB inv bit (18) set

- **[DevSNB:A{W/A}]** If bit 13 of MI_MODE is ‘0’, pipelined PIPE_CONTROLs cannot be between multiple non-pipelined state if there are no 3DPRIMITIVE commands previously. Legal and illegal examples below

Ring/Batch Contents - ILLEGAL
3DPRIMITIVE
np-state
pipelined (bit 20 = ‘0’) PIPE_CONTROL
np-state
3DPRIMITIVE

Ring/Batch Contents - LEGAL
3DPRIMITIVE
np-state
3DPRIMITIVE
pipelined (bit 20 = ‘0’) PIPE_CONTROL

Ring/Batch Contents - LEGAL
np-state
3DPRIMITIVE

- [DevSNB-A{W/A}] For all PIPE_CONTROLs that *only* have RO cache invalidation, software must set the post-sync operation field to something other than 0

- [DevSNB-A {W/A}] For all PIPE_CONTROLs that has **Stall At Pixel Scoreboard** set, software must also set either the **Depth Stall** bit or the **CS Stall** bit.

- DevSNB A {W/A}] [DevSNB B {W/A}] Before any depth stall flush (including those produced by non-pipelined state commands), software needs to first send a PIPE_CONTROL with the **CS Stall** bit set. **CS Stall** restrictions still apply, *except* for setting the **Depth Stall** bit. This bit cannot be set on this PIPE_CONTROL.

- [DevSNB-C+{W/A}] Before any depth stall flush (including those produced by non-pipelined state commands), software needs to first send a PIPE_CONTROL with no bits set *except* **Post-Sync Operation** != 0.

- [Dev-SNB{W/A}]: Before a PIPE_CONTROL with **Write Cache Flush Enable =1,** a PIPE_CONTROL with any non-zero post-sync-op is required.

- Dev-SNB{W/A}]: Pipe-control with CS-stall bit set must be sent BEFORE the pipe-control with a post-sync op and no write-cache flushes.

Table 9. Caches Invalidated/Flushed by PIPE_CONTROL Bit Settings

The table below explains all the different flush/invalidation scenerios for DevSNB+

Write cache flush	Notification Enabled	non-VF RO Cache Invalidate	VF RO Cache Invalidate	Marker Sent	pipeline marker enable	Completion Requested	Top of pipe invalidate pulse from CS
0	0	0	0	N/A	N/A	N/A	N/A
0	0	0	1	Yes	No	N/A	No
0	0	1	0	No	N/A	N/A	Yes
0	0	1	1	Yes	No	No	Yes
X	1	0	X	Yes	Yes	Yes	No
X	1	1	X	Yes	Yes	Yes	Yes
1	X	0	X	Yes	Yes	Yes	No
1	X	1	X	Yes	Yes	Yes	Yes

PIPE_CONTROL

Project:	DevSNB+	**Length Bias:**	2

The PIPE_CONTROL command is used to effect the synchronization described above.

DWord	Bit	Description
0	31:29	Command Type Default Value: 3h GFXPIPE Format: OpCode
	28:27	Command SubType Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	3D Command Opcode Default Value: 2h PIPE_CONTROL Format: OpCode
	23:16	3D Command Sub Opcode Default Value: 0h PIPE_CONTROL Format: OpCode
	15:8	Reserved Project: All Format: MBZ
	7:0	DWord Length Default Value: 2h 3h for QWord Write Format: =n Total Length - 2 Project: All
1	31:26	Reserved Format: MBZ

PIPE_CONTROL		
	25	Reserved Format: MBZ
	24	Reserved Project: DevSNB Format: MBZ
	23	Reserved
	22	Protected Memory Enable Project: All Format: U1 After completion of the flush, the hardware will limit all access to the Protected Content Memory. Only command streamer initiated cacheable writes are allowed to non-PCM memory. Programming Note: This bit is ignored if only read-only invalidate bits are set (no write flush, depth stall, or post-sync op) [DevSNB D+] Once set, it can only be cleared at the end of a batch buffer with **Clear Command Buffer Enable** bit set. This applies to soft resets also, including FLR [pre-DevSNB D] This bit is always set/cleared by the PIPE_CONTROL command with write flush, depth stall, or post-sync op set.
	21	Store Data Index Project: All Format: U1 This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is actually an index into the hardware status page. This bit only applies to the Global HW status page. If this field is set to '1', the **Destination Address Type** in this command must be set to '1' (GGTT) If this bit is set, this command will index into the per-process hardware status page if executed from within a non-secure batch buffer and if the **Per-Process Virtual Address Space** bit is set. Else the Global HW status page is used.
	20	CS Stall Project: All Format: U1 If ENABLED, the sync operation will not occur until all previous flush operations pending a completion of those previous flushes will complete, including the flush produced from this command. This enables the command to act similar to the legacy MI_FLUSH command.

PIPE_CONTROL

19 — Global Snapshot Count Reset

Project: All

Format: U1

SW should never set this bit during normal operation since the Statistics Counters are intended to be free running.

Value	Name	Description	Project
0h	Don't Reset	Do not reset the snapshot counts or Statistics Counters.	All
1h	Reset	Reset the snapshot count for all the units and reset the Statistics Counters except as noted above.	All

Programming Notes
PS_DEPTH_COUNT and TIMESTAMP are *not* reset by PIPE_CONTROL with this bit set. TIMESTAMP and PS_DEPTH_COUNT can be reset by writing 0 to them

18 — TLB Invalidate Project: All Format: U1

If ENABLED, all TLBs will be invalidated once the flush operation is complete. Note that if the flush TLB invalidation mode is clear, a TLB invalidate will occur irrespective of this bit setting

17 — Synchronize GFDT Surface Project: All Format: U1

If enabled, at the end of the current flush the last level cache is cleared of all the cachelines which have been marked with the special GFDT flags. Store DW must be enabled.

16 — Generic Media State Clear Project: DevSNB+ Format: Disable

If set, all generic media state context information will not be included with the next context save, assuming no new state is initiated after the flush. If clear, the generic media state context save state will not be affected. An MI_FLUSH with this bit set should be issued once all the Media Objects that will be processed by a given persistent root thread have been issued or when an MI_SET_CONTEXT switching from a generic media context to a 3D context completes. When using MI_SET_CONTEXT, once state is programmed, it will be saved and restarted as part of any context each time that context is saved/restored until an MI_FLUSH with this bit set is issued in that context.

15:14 Post-Sync Operation

Project: All

This field must be cleared if the **LRI Post-Sync Operation** bit is set.

Value	Name	Description	Project
0h	No Write	No write occurs as a result of this instruction. This can be used to implement a "trap" operation, etc.	All
1h	Write Immediate Data	Write the QWord containing Immediate Data Low, High DWs to the Destination Address	All
2h	Write PS Depth Count	Write the 64-bit PS_DEPTH_COUNT register to the Destination Address	All
3h	Write Timestamp	Write the 64-bit TIMESTAMP register to the Destination Address	All

Programming Notes
If executed in non-secure batch buffer, the address given will be in a PPGTT address space. If in a secure ring or batch, address given will be in GGTT space

13 Depth Stall Enable

Project: All

Format: Enable

This bit should be set when obtaining a "visible pixel" count to preclude the possible inclusion in the PS_DEPTH_COUNT value written to memory of some fraction of pixels from objects initiated *after* the PIPE_CONTROL command.

Value	Name	Description	Project
0h	Disable	3D pipeline will not stall subsequent primitives at the Depth Test stage.	All
1h	Enable	3D pipeline will stall any subsequent primitives at the Depth Test stage until the Sync and Post-Sync operations complete.	All

Programming Notes
This bit should be DISABLED for operations other than writing PS_DEPTH_COUNT.
This bit will have no effect (besides preventing write cache flush) if set in a PIPE_CONTROL command issued to the Media pipe.

PIPE_CONTROL

12 — Render Target Cache Flush Enable

Project: All

Format: Enable

Setting this bit will force Render Cache to be flushed to memory prior to this synchronization point completing. This bit should be set for all write fence sync operations to assure that results from operations initiated prior to this command are visible in memory once software observes this synchronization.

Value	Name	Description	Project
0h	Disable Flush	Render Target Cache is NOT flushed.	All
1h	Enable Flush	Render Target Cache is flushed.	All

Programming Notes
This bit should be DISABLED for End-of-pipe (Read) fences, PS_DEPTH_COUNT or TIMESTAMP queries.
This bit must not be set when Depth Stall Enable bit is set in this packet.

11 — Instruction Cache Invalidate Enable — Project: All — Format: Enable

Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the L1 and L2 at the top of the pipe i.e. at the parsing time.

10 — Texture Cache Invalidation Enable — Project: All — Format: Enable

Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the texture caches at the top of the pipe i.e. at the parsing time.

9 — Indirect State Pointers Disable — Project: All — Format: Enable

At the completion of the post-sync operation associated with this pipecontrol packet, the indirect state pointers in the hardware will be considered as invalid ie the indirect pointers will not be saved in the context. If any new indirect state commands are executed in the command stream while the pipe control is pending, the new indirect state commands will be preserved.

[pre-DevSNB C] It is considered UNDEFINED if there is a PIPE_CONTROL with this bit set *before* any pipelined state in any context is set. For instance, it is not allowed to send a PIPE_CONTROL with this bit set as a first command coming out of reset.

[pre-DevSNB C] It is considered UNDEFINED to have more than 1 PIPE_CONTROL with this bit set for every 1 configuration of pipeline state.

PIPE_CONTROL		
	8	Notify Enable Project: All Format: Enable If ENABLED, a Sync Completion Interrupt will be generated (if enabled by the MI Interrupt Control registers) once the sync operation is complete. See Interrupt Control Registers in *Memory Interface Registers* for details.
	7	Reserved Project: Format:
	6	Protected Memory Application ID Project: DevSNB+ Default Value: 0h DefaultVaueDesc Format: U1 FormatDesc Context ID for Media Applications
	5	Reserved Project: Format:
	4	VF(address based) Cache Invalidation Enable Project: All Format: Enable Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of VF address based cache at the top of the pipe i.e. at the parsing time.
	3	Constant Cache Invalidation Enable Project: All Format: Enable Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the constant cache at the top of the pipe i.e. at the parsing time.
	2	State Cache Invalidation Enable Project: All Format: Enable Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the L1 and L2 state caches at the top of the pipe i.e. at the parsing time.

PIPE_CONTROL

1 — Stall At Pixel Scoreboard

Project: All

Format: Enable

Defines the behavior of PIPE_CONTROL command at the pixel scoreboard.

Value	Name	Description	Project
0h	Disable	Stall at the pixel scoreboard is disabled.	All
1h	Enable	Stall at the pixel scoreboard is enabled.	All

Programming Notes
This bit should be DISABLED for End-of-pipe (Read) fences, PS_DEPTH_COUNT or TIMESTAMP queries. This bit is ignored if Depth Stall Enable is set. Further the render cache is not flushed even if Write Cache Flush Enable bit is set.

0 — Depth Cache Flush Enable

Project: All

Format: Enable

Setting this bit enables flushing (i.e. writing back the dirty lines to memory and invalidating the tags) of depth related caches. This bit applies to HiZ cache, Stencil cache and depth cache.

Value	Name	Description	Project
0h	Flush Disabled	Depth relates caches (HiZ, Stencil and Depth) are NOT flushed.	All
1h	Flush Enabled	Depth relates caches (HiZ, Stencil and Depth) are flushed.	All

Programming Notes
Ideally depth caches need to be flushed only when depth is required to be coherent in memory for later use as a texture, source or honoring CPU lock. This bit should be DISABLED for End-of-pipe (Read) fences, PS_DEPTH_COUNT or TIMESTAMP queries.
This bit must not be set when Depth Stall Enable bit is set in this packet.

PIPE_CONTROL

2 — 31:3

Address

Project: DevSNB

Address: GraphicsAddress[31:3]

Surface Type: U32

This field specifies Bits 31:3 of the Address where the DWord or QWord will be stored. As the store address must be DWord-aligned, Bits 1:0 of that address MBZ. This address must be 8B aligned for a store "QW" command.

2 — 2

Destination Address Type

Project: DevSNB

Defines address space of Destination Address

Value	Name	Description	Project
0h	Reserved		
1h	GGTT	Use GGTT address space for DW write	All

Programming Notes
Ignored if "No write" is the selected in Operation.

2 — 1:0

Reserved Project: All Format: MBZ

3..4 — 31:0

Immediate Data

Project: All

Format: U32

This field specifies the DWord value to be written to the targeted location. DW3 is the lower DW. Only valid when **LRI Post-Sync Operation** is set.

1.7.4.2 [DevSNB+] Programming Restrictions for PIPE_CONTROL

PIPE_CONTROL arguments can be split up into three categories:

- Post-sync operations
- Flush Types
- Stall

Post-sync operation is only indirectly affected by the flush type category via the stall bit. The stall category depends on the both flush type and post-sync operation arguments. A PIPE_CONTROL with no arguments set is **Invalid**

1.7.4.2.1 Post-Sync Operation

These are arguments related to events that occur *after* the marker initiated by the PIPE_CONTROL command is completed. The table below shows the restrictions:

Arguments	Bit	Restrictions
Protected Mem Enable	22	Requires stall bit ([20] of DW1) set.
Global Snapshot Count Reset	19	Requires stall bit ([20] of DW1) set.
Generic Media State Clear	16	Requires stall bit ([20] of DW1) set.
Indirect State Pointers Disable	9	Requires stall bit ([20] of DW1) set.
Store Data Index	21	Post-Sync Operation ([15:14] of DW1) must be set to something other than ‘0’
Sync GFDT	17	Post-Sync Operation ([15:14] of DW1) must be set to something other than ‘0’ or 0x2520[13] must be set
TLB inv	18	Post-Sync Operation ([15:14] of DW1) must be set to something other than ‘0’. Already implied when 0x2520[13] is set
Post Sync Op	15:14	No Restriction
Notify En	8	No Restriction

1.7.4.2.2 Flush Types

These are arguments related to the type of read only invalidation or write cache flushing is being requested. Note that there is only intra-dependency. That is, it is not affected by the post-sync operation or the stall bit. The table below shows the restrictions:

Arguments	Bit	Restrictions
Depth Stall	13	Following bits must be *clear* • Render Target Cache Flush Enable ([12] of DW1) • Depth Cache Flush Enable ([0] of DW1)
Render Target Cache Flush	12	Depth Stall must be clear ([13] of DW1)
Depth Cache Flush	0	Depth Stall must be clear ([13] of DW1)
Stall Pixel Scoreboard	1	No Restriction
Inst invalidate.	11	No Restriction
Tex invalidate.	10	No Restriction
VF invalidate	4	No Restriction
Constant invalidate	3	No Restriction
State Invalidate	2	No Restriction

1.7.4.2.3 Stall

If the stall bit is set, the command streamer waits until the pipe is completely flushed.

Arguments	Bit	Restrictions
Stall Bit	20	1 of the following must also be set • Render Target Cache Flush Enable ([12] of DW1) • Depth Cache Flush Enable ([0] of DW1) • Stall at Pixel Scoreboard ([1] of DW1) • Depth Stall ([13] of DW1) • Post-Sync Operation ([13] of DW1) • Notify Enable ([8] of DW1)

2. Vertex Fetch (VF) Stage

2.1 Vertex Fetch (VF) Stage Overview

The VF stage performs one major function: executing 3DPRIMITIVE commands. This is handled by the VF's InputAssembly function. Minor enhancements have been included to better support legacy D3D APIs as well as OpenGL.

The following subsections describe some high-level concepts associated with the VF stage.

2.1.1 Input Assembly

The VF's InputAssembly function includes (for each vertex generated):

- Generation of VertexIndex and InstanceIndex for each vertex, possibly via use of an Index Buffer.
- Lookup of the VertexIndex in the Vertex Cache (if enabled)
- If a cache miss is detected:
    - Use of computed indices to fetch data from memory-resident vertex buffers
    - Format conversion of the fetched vertex data
    - Assembly of the format conversion results (and possibly some internally generated data) to form the complete "input" (raw) vertex
    - Storing the input vertex data in a Vertex URB Entry (VUE) in the URB
    - Output of the VUE handle of the input vertex to the VS stage
- If a cache hit is detected, the VUE handle from the Vertex Cache is passed to the VS stage (marked as a cache hit to prevent any VS processing).

2.1.1.1 Vertex Assembly

The VF utilizes a number of VERTEX_ELEMENT state structures to define the contents and format of the vertex data to be stored in Vertex URB Entries (VUEs) in the URB. See below for a detailed description of the command used to define these structures (3DSTATE_VERTEX_ELEMENTS).

Each active VERTEX_ELEMENT structure defines up to 4 contiguous DWords of VUE data, where each DWord is considered a "component" of the vertex element. The starting destination DWord offset of the vertex element in the VUE is specified, and the VERTEX_ELEMENT structures must be defined with monotonically increasing VUE offsets. For each component, the source of the component is specified. The source may be a constant (0, 0x1, or 1.0f), a generated ID (VertexID, InstanceID or PrimitiveID), or a

component of a structure in memory (e.g,. the Y component of an XYZW position in memory). In the case of a memory source, the Vertex Buffer sourcing the data, and the location and format of the source data with that VB are specified.

The VF's Vertex Assembly process can be envisioned as the VF unit stepping through the VERTEX_ELEMENT structures in order, fetching and format-converting the source information (if memory resident), and storing the results in the destination VUE.

2.1.2 Vertex Cache

The VF stage communicates with the VS stage in order to implement a Vertex Cache function in the 3D pipeline. The Vertex Cache is strictly a performance-enhancing feature and has no impact on 3D pipeline results (other than a few statistics counters).

The Vertex Cache contains the VUE handles of VS-output (shaded) vertices if the VS function is enabled, and the VUE handles of VF-output (raw) vertices if the VS function is disabled. (Note that the actual vertex data is held in the URB, and only the handles of the vertices are stored in the cache). In either case, the contents of the cache (VUE handles) are tagged with the VertexIndex value used to fetch the input vertex data. The rationale for using the VertexIndex as the tag is that (assuming no other state or parameters change) a vertex with the same VertexIndex as a previous vertex will have the same input data, and therefore the same result from the VF+VS function.

Note that any change to the state controlling the InputAssembly function (e.g., vertex buffer definition), or any change to the state controlling the VS function (if enabled) (e.g., VS kernel), will result in the Vertex Cache being invalidated. In addition, any non-trivial use of instancing (i.e., more than one instance per 3DPRIMITIVE command and the inclusion of instance data in the input vertex) will effectively invalidate the cache between instances, as the InstanceIndex is not included in the cache tag. See Vertex Caching in *Vertex Shader* for more information on the Vertex Cache (e.g., when it is implicitly disabled, etc.)

2.1.3 Input Data: Push Model vs. Pull Model

Given the programmability of the pipeline, and the ability of shaders to input (load/sample) data from memory buffers in an arbitrary fashion, the decision arises in whether to push instance/vertex data into the front of the pipeline or defer the data access (pull) to the shaders that require it.

There are tradeoffs involved in deciding between these models. For vertex data, it is probably always better to push the data into the pipeline, as the VF hardware attempts to cover the latency of the data fetch. The decision is less clear for instance data, as pushing instance data leads to larger Vertex URB entries which will be holding redundant data (as the instance data for vertices of an object are by definition the same). Regardless, the 3D pipeline supports both models.

2.1.4 Generated IDs

[Note that the generated IDs are considered separate from any offset computations performed by the VF unit, and are therefore described separately here.]

The VF generates InstanceID, VertexID, and PrimitiveID values as part of the InputAssembly process.

VertexID and InstanceID are only allowed to be inserted into the input vertex data as it is gathered and written into the URB as a VUE.

The definition/use of PrimitiveID is more complicated than the other auto-generated IDs. PrimitiveID is associated with an “object”, not a particular vertex. It is only available to the GS as a special non-vertex input, and the PS as a constant-interpolated attribute. It is not seen by the VS at all. The PrimitiveID therefore is kept separate from the vertex data. Take for example a TRILIST primitive topology: It should be possible to share vertices between triangles in the list (i.e., reuse the VS output of a vertex), even though each triangle has a different PrimitiveID associated with it.

2.1.4.1 Generated IDs [DevSNB]

The InstanceID, VertexID, and PrimitiveID values associated with each vertex can be stored in the vertex’s VUE, via use of the **Component *n* Control** fields in the VERTEX_ELEMENT structure. This makes the value(s) available to the VS thread.

[**DevSNB+**]: While the PrimitiveID can still be stored in the VUE (see above), there should be no API-specific reason to do so. The 32-bit PrimitiveIDs associated with objects are passed down the FF pipeline and made available to GS and Setup threads as payload header data. A side effect of this feature is that the vertex cache can operate even when PrimitiveIDs are being used.

2.2 Index Buffer (IB)

The 3DSTATE_INDEX_BUFFER command is used to define an *Index Buffer* (IB) used in subsequent 3DPRIMITIVE commands.

The RANDOM access mode of the 3DPRIMITIVE command involves the use of a memory-resident IB. The IB, defined via the 3DSTATE_INDEX_BUFFER command described below, contains a 1D array of 8, 16 or 32-bit index values. These index values will be fetched by the InputAssembly function, and subsequently used to compute locations in VERTEXDATA buffers from which the actual vertex data is to be fetched. (This is opposed to the SEQUENTIAL access mode were the vertex data is simply fetched sequentially from the buffers).

Software is responsible for ensuring that accesses outside the IB do not occur. This is possible as software can compute the range of IB values referenced by a 3DPRIMITIVE command (knowing the **StartVertexLocation**, **InstanceCount**, and **VerticesPerInstance** values) and can then compare this range to the IB extent.

2.2.1 3DSTATE_INDEX_BUFFER [DevSNB+]

3DSTATE_INDEX_BUFFER			
Project:	**All**	**Length Bias:**	**2**

This command is used to specify the current IB state used by the VF function. At most one IB is defined and active at any given time.

NOTES:

The IB must be specified before any RANDOM 3D_PRIMITIVE commands are issued

It is possible to have vertex elements source completely from generated ID values and therefore not require any Index Buffer accesses. In this case, VF function will simply ignore the Index Buffer state.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Ah 3DSTATE_INDEX_BUFFER Format: OpCode
	15:12	**Index Buffer Object Control State** Project: [DevSNB+] Format: MEMORY_OBJECT_CONTROL_STATE FormatDesc Specifies the memory object control state for this index buffer.
	11	**Reserved** Project: All Format: MBZ

3DSTATE_INDEX_BUFFER

	10	**Cut Index Enable** Project: [DevSNB+] Format: Enable FormatDesc If ENABLED, the largest index value (0xFF,0xFFFF,0xFFFFFFFF, depending on Index Format) is interpreted as the “cut” index. (See description of this elsewhere in this section). If DISABLED, there is no special “cut” index value, and the largest index value is simply used as an index. (Expected OpenGL driver usage) This field can only be enabled for certain primitive topology types. Refer to the table later in this section for details.
	9:8	**Index Format** Project: All Format: U2 enumerated type FormatDesc This field specifies the data format of the index buffer. All index values are UNSIGNED.

Value	Name	Description	Project
0h	INDEX_BYTE		All
1h	INDEX_WORD		All
2h	INDEX_DWORD		All

	7:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All

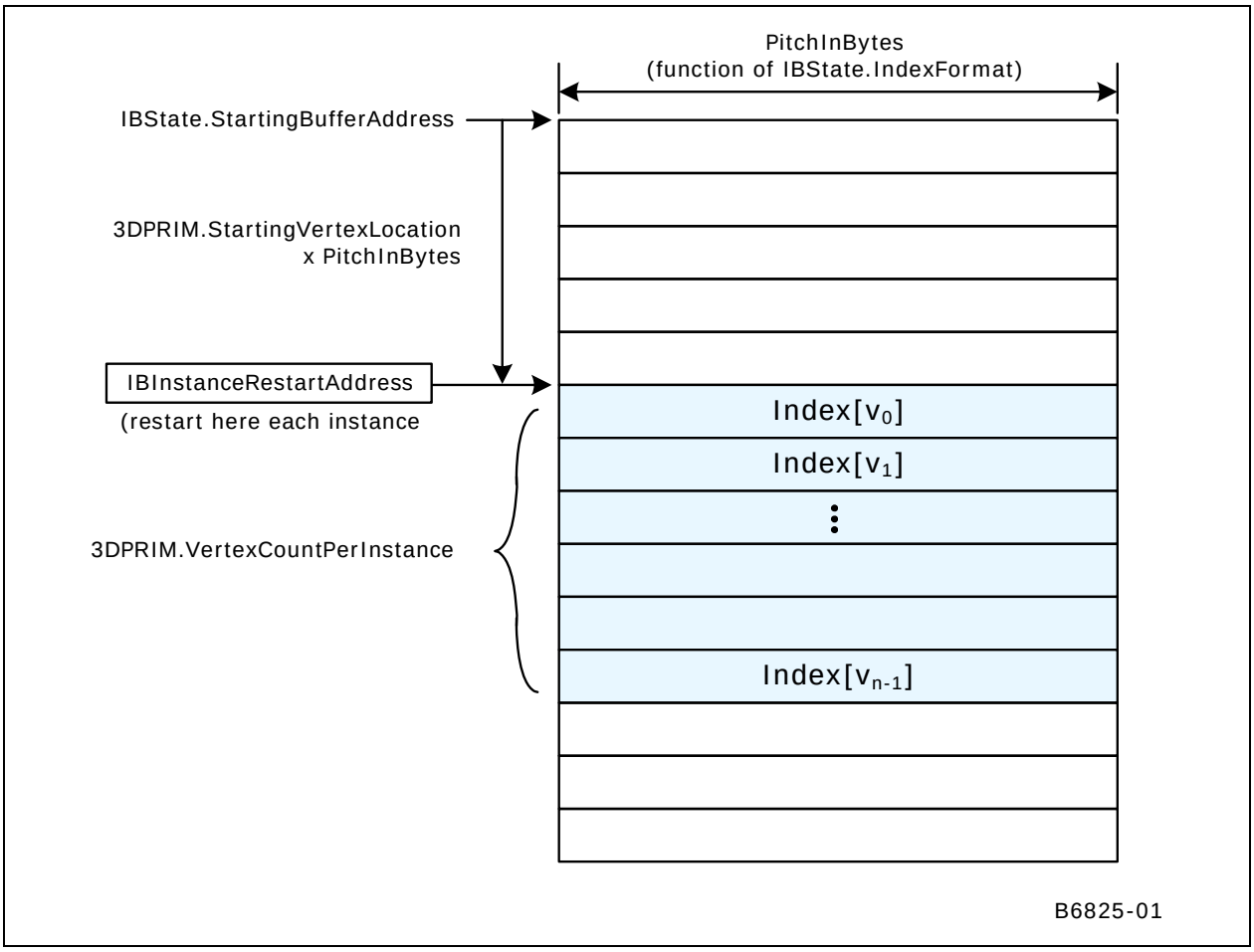
3DSTATE_INDEX_BUFFER		
1	31:0	**Buffer Starting Address** Project: All Address: GraphicsAddress[31:0] Surface Type: Index Buffer Entry This field contains the size-aligned (as specified by Index Format) Graphics Address of the first element of interest within the index buffer. Software must program this value with the combination (sum) of the base address of the memory resource and the byte offset from the base address to the starting structure within the buffer. **Programming Notes** Index Buffers can only be allocated in linear (not tiled) graphics memory
2	31:0	**Buffer Ending Address** Project: All Address: GraphicsAddress[31:0] If non-zero, this field contains the address of the last valid byte in the index buffer. Any index buffer reads past this address returns an index value of 0 (as if the index buffer was zero-extended). Software must guarantee that the buffer ends on an index boundary (e.g., for an INDEX_DWORD buffer, Bits [1:0] == 11b).

The following table lists which primitive topology types support the presence of Cut Indices. When 3DSTATE_INDEX_BUFFER has **Cut Index Enable** set, it is UNDEFINED to issue a 3DPRIMITIVE with a primitive topology type not supporting a Cut Index (even if no cut indicies are actually present in the index buffer).

Definition	Cut Index?
3DPRIM_POINTLIST	Y
3DPRIM_LINELIST	Y
3DPRIM_LINESTRIP	Y
3DPRIM_TRILIST	Y
3DPRIM_TRISTRIP	Y
3DPRIM_TRIFAN	DevSNB: N
3DPRIM_QUADLIST	DevSNB: N
3DPRIM_QUADSTRIP	DevSNB: N
3DPRIM_LINELIST_ADJ	Y
3DPRIM_LINESTRIP_ADJ	Y
3DPRIM_TRILIST_ADJ	Y
3DPRIM_TRISTRIP_ADJ	Y
3DPRIM_TRISTRIP_REVERSE	Y
3DPRIM_POLYGON	DevSNB: N
3DPRIM_RECTLIST	N
3DPRIM_LINELOOP	DevSNB: N
3DPRIM_POINTLIST_BF	Y
3DPRIM_LINESTRIP_CONT	Y
3DPRIM_LINESTRIP_BF	Y
3DPRIM_LINESTRIP_CONT_BF	Y
3DPRIM_TRIFAN_NOSTIPPLE	N
3DPRIM_PATCHLIST_n	DevSNB: N

2.2.2 Index Buffer Access

The figure below illustrates how the Index Buffer is accessed.



2.3 Vertex Buffers (VBs)

The 3DSTATE_VERTEX_BUFFERs and 3DSTATE_INSTANCE_STEP_RATE commands are used to define *Vertex Buffers* (VBs) used in subsequent 3DPRIMITIVE commands.

Most input vertex data is sourced from memory-resident VBs. A VB is a 1D arrays of structures, where the size of the structure as defined by the VB's **BufferPitch**. VBs are accessed either as *VERTEXDATA buffers* or *INSTANCEDATA buffers*, as defined by the VB's **BufferAccessType**. The VB's access type will determine whether the VF-computed VertexIndex or InstanceIndex is used to access data in the VB.

Given that the RANDOM access mode of the 3DPRIMITIVE command utilizes an IB (possibly provided by an application) to compute VB index values, VB definitions contain a **MaxIndex** value used to detect accesses beyond the end of the VBs. Any access outside the extent of a VB returns 0.

2.3.1 3DSTATE_VERTEX_BUFFERS

This command is used to specify VB state used by the VF function. From 1 to 33 VBs can be specified, where the **VertexBufferID** field within the VERTEX_BUFFER_STATE structure(s) indicate the specific VB. If a VB definition is not included in this command, its associated state is left unchanged and available for use if previously defined.

NOTES:

- It is possible to have individual vertex elements sourced completely from generated ID values and therefore not require any vertex buffer accesses for that vertex element. In this case, VF function will simply ignore the VB state associated with that vertex element. If all enabled vertex elements have this characteristic, no VBs are required to process 3DPRIMITIVE commands. For example, this might arise when the user wants to perform all data lookups in the first shader, so only generated index values need to be passed down to it. In this extreme case, SW would not need to program any VB state, and therefore not need to issue any 3DSTATE_VERTEX_BUFFERS commands.

- For any 3DSTATE_VERTEX_BUFFERS command, at least one VERTEX_BUFFER_STATE structure must be included.

- VERTEX_BUFFER_STATE structures are 4 DWords for both VERTEXDATA buffers and INSTANCEDATA buffers.

- Inclusion of partial VERTEX_BUFFER_STATE structures is UNDEFINED.

The order in which VBs are defined within this command can be arbitrary, though a vertex buffer must be defined only once in any given command (otherwise operation is UNDEFINED).

DWord	Bit	Description
0	31:29	**Command Type =** GFXPIPE = 03h
	28:16	**GFXPIPE Opcode =** 3DSTATE_VERTEX_BUFFERS GFXPIPE[28:27 = 3h, 26:24 = 0h, 23:16 = 08h] (Pipelined)
	15:8	Reserved : MBZ
	7:0	**DWord Length** (excludes DWords 0,1) 4n-1 (where n = # of buffer states included)
1-4		**Vertex Buffer State [0]** Format: VERTEX_BUFFER_STATE
5-8		**Vertex Buffer State [1]**
…		…
(4n-3)-(4n)		**Vertex Buffer State [..]**

2.3.2 VERTEX_BUFFER_STATE Structure

2.3.2.1 VERTEX_BUFFER_STATE Structure [DevSNB]

VERTEX_BUFFER_STATE	
Project:	DevSNB

This structure is used in 3DSTATE_VERTEX_BUFFERS to set the state associated with a VB. The VF function will use this state to determine how/where to extract vertex element data for all vertex elements associated with the VB.

The VERTEX_BUFFER_STATE structure is 4 DWords for both INSTANCEDATA and VERTEXDATA buffers.

A VB is defined as a 1D array of vertex data structures, accessed via a computed index value. The VF function therefore needs to know the starting address of the first structure (index 0) and size of the vertex data structure. [DevILK+] Vertex element accesses which straddle or go past the VB's End Address will return 0's for all elements.

VERTEX_BUFFER_STATE

DWord	Bit	Description
0	31:26	**Vertex Buffer Index** Project: [DevSNB+] Format: U6 index FormatDesc Address: GraphicsAddress[31:0] Range [0,32] This field contains an index value which selects the VB state being defined.
	25:21	**Reserved** Project: All Format: MBZ
	20	Buffer Access Type This field determines how vertex element data is extracted from this VB. This control applies to all vertex elements associated with this VB.

Value	Name	Description	Project
00	VERTEXDATA	For SEQUENTIAL vertex access, each vertex of an instance is sourced from sequential structures within the VB. For RANDOM vertex access, each vertex of an instance is looked up (separately) via a computed index value	All
01	INSTANCEDATA	Each vertex of an instance is sourced with the same (instance) data. Subsequent instances may be sourced with the same or different data, depending on **Instance Data Step Rate**.	All

19:16	Vertex Buffer Memory Object Control State Project All Format: MEMORY_OBJECT_CONTROL_STATE FormatDesc Specifies the memory object control state for this vertex buffer.	

VERTEX_BUFFER_STATE		
	15	Reserved Project: All Format: MBZ
	14	Reserved: MBZ
	13	Null Vertex Buffer. Project: [DevILK+] Format: Enable FormatDesc This field enabled causes any fetch for vertex data to return 0.
	12	bitfieldname Project: [DevCTG+] Security: None Access: None Exists If: Always Default Value: 0h DefaultVaueDesc Mask: MMIO(0x2000)#16 Format: U32 FormatDesc Address: GraphicsAddress[31:0] Surface Type: U32 Range 0..2^32-1 Invalidate the Vertex overfetch cache when this bit is set. For multiple vertex buffer state structures in one packet, this bit may be set only once in the entire packet. [Pre-DevCTG]: Reserved

VERTEX_BUFFER_STATE

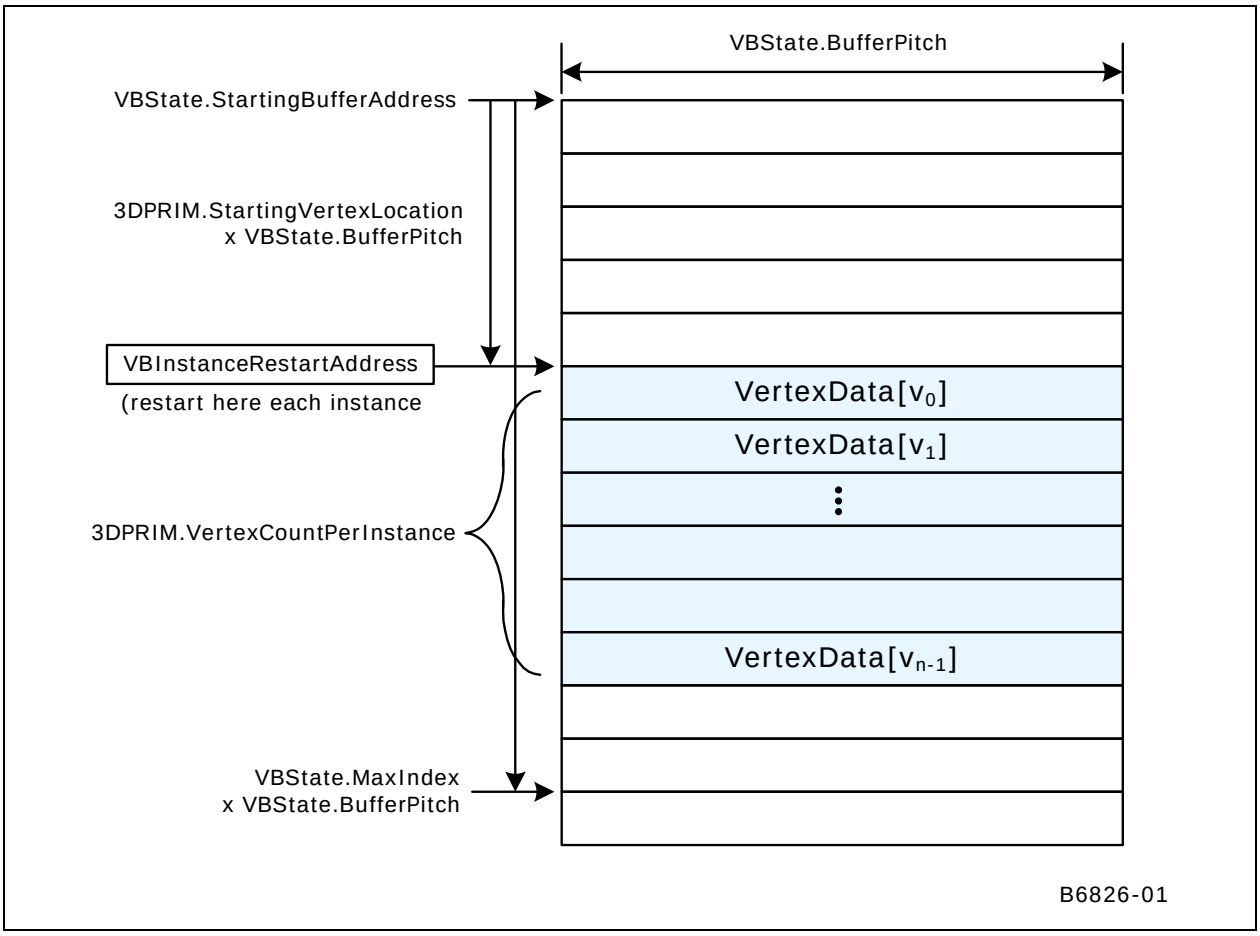
	11:0	Buffer Pitch Format: U12 — FormatDesc Count of bytes Range [Pre-DevCTG]: [0,2047] Bytes [DevCTG+]: [0,2048] Bytes This field specifies the pitch in bytes of the structures accessed within the VB. This information is required in order to access elements in the VB via a structure index. Programming Notes Different VERTEX_BUFFER_STATE structures can refer to the same memory region using different Buffer Pitch values. See note on 64-bit float alignment in Buffer Starting Address.
1	31:0	**Buffer Starting Address** Format: GraphicsAddress[31:0] — FormatDesc Address: GraphicsAddress[31:0] This field contains the byte-aligned Graphics Address of the first element of interest within the VB. Software must program this value with the combination (sum) of the base address of the memory resource and the byte offset from the base address to the starting structure within the buffer. **Programming Notes** 64-bit floating point values must be 64-bit aligned in memory, or UNPREDICTABLE data will be fetched. When accessing an element containing 64-bit floating point values, the Buffer Starting Address and Source Element Offset values must add to a 64-bit aligned address, and BufferPitch must be a multiple of 64-bits. VBs can only be allocated in linear (not tiled) graphics memory. As computed index values are, by definition, interpreted as unsigned values, there is no issue with accesses to locations before (lower address value) the start of the buffer. However, these wrapped indices are subject to Max Index checking (see below).

VERTEX_BUFFER_STATE		
2	31:0	**End Address** Project: All Security: None Default Value: 0h DefaultVaueDesc Mask: MMIO(0x2000)#16 Format: U32 FormatDesc Address: GraphicsAddress[31:0] Surface Type: U32 Range 0..2^32-1 [DevIL+] This field defines the address of the last valid byte in this particular VB. Access of a vertex element which either straddles or is beyond this address will return 0's for any data read.
2	31:0	**Max Index [Pre-DevILK]** Format: U32 FormatDesc If non-zero (bounds-checking enabled), this field defines the maximum (inclusive) structure index accessible for this particular VB. Use of an index larger than the Max Index returns 0 for all components. This includes a "negative" computed index which, when viewed as an unsigned value, exceeds Max Index. If zero, bounds checking is disabled. A read from the vertex buffer memory is performed regardless of the computed index. **Programming Notes** The smallest vertex buffer that can be bounds-checked is a 2-entry buffer (where MaxIndex is programmed to 1).

VERTEX_BUFFER_STATE		
3	31:0	**Instance Data Step Rate:** Format: U32 FormatDesc This field only applies to INSTANCEDATA buffers – it is ignored (but still present) for VERTEXDATA buffers). This field determines the rate at which instance data for this particular INSTANCEDATA vertex buffer is changed in sequential instances. Only after the number of instances specified by this field is generated is new (sequential) instance data provided. This process continues for each group of instances defined in the draw command. For example, a value of 1 in this field causes new instance data to be supplied with each sequential (instance) group of vertices. A value of 2 causes every other instance group of vertices to be provided with new instance data. The special value of 0 causes all vertices of all instances generated by the draw command to be provided with the same instance data. (The same effect can be achieved by setting this field to its maximum value.)

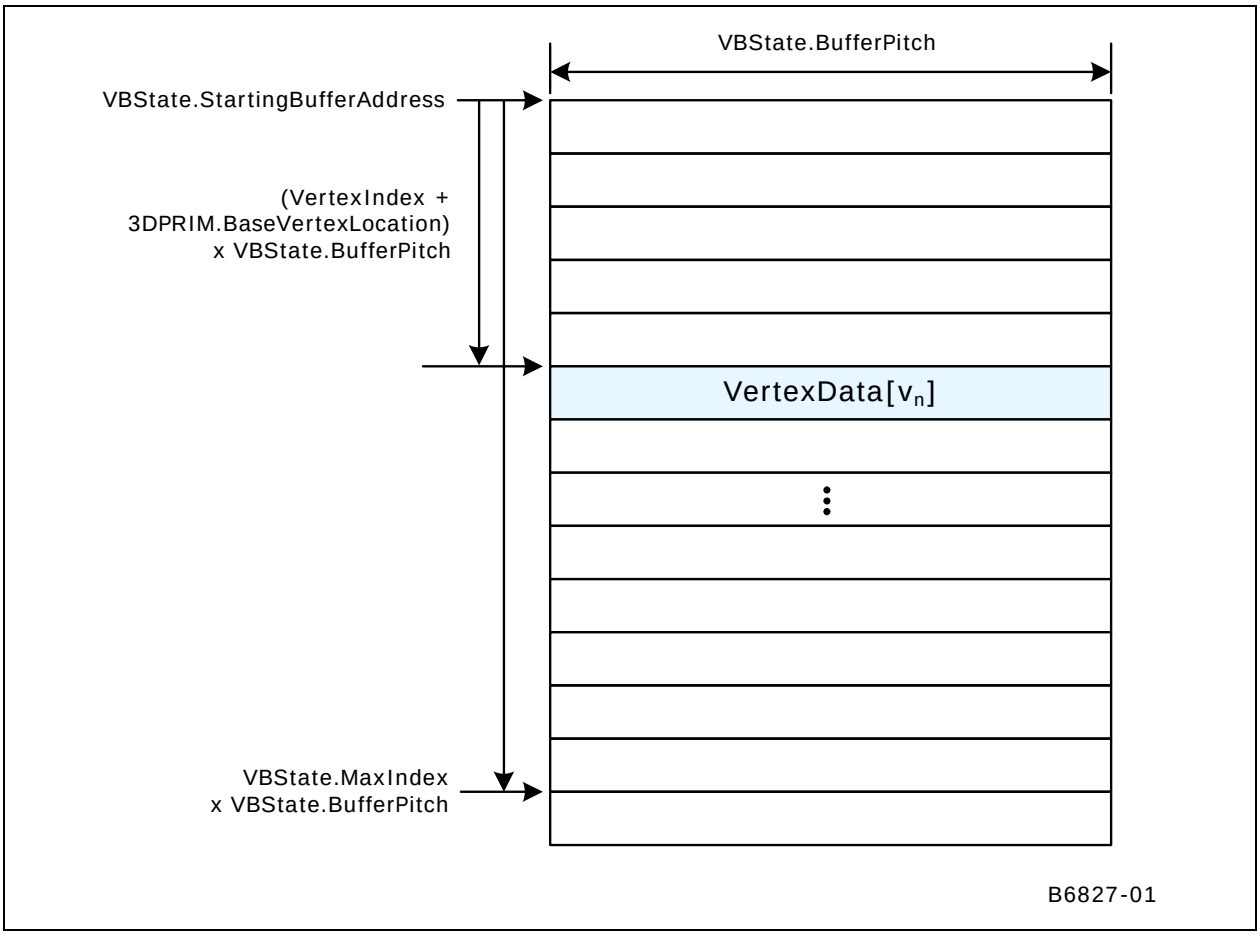
2.3.3 VERTEXDATA Buffers – SEQUENTIAL Access

Instead of “VBState.StartingBufferAddress + VBState.MaxIndex x VBState.BufferPitch”, the address of the byte immediately beyond the last valid byte of the buffer is determined by “VBState.EndAddress+1”.



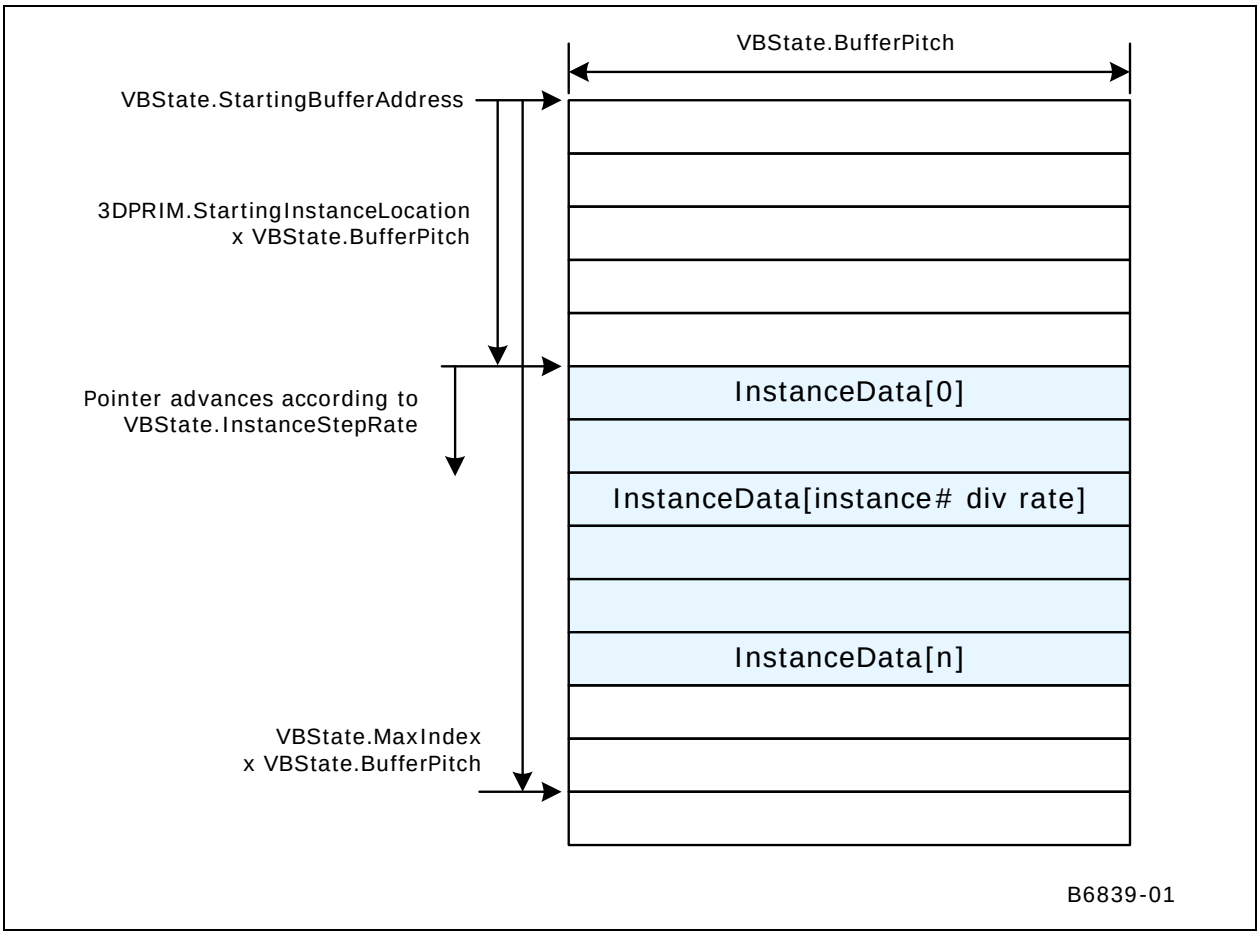
2.3.4 VERTEXDATA Buffers – RANDOM Access

Instead of “VBState.StartingBufferAddress + VBState.MaxIndex x VBState.BufferPitch”, the address of the byte immediately beyond the last valid byte of the buffer is determined by “VBState.EndAddress+1”.



2.3.5 INSTANCEDATA Buffers

Instead of “VBState.StartingBufferAddress + VBState.MaxIndex x VBState.BufferPitch”, the address of the byte immediately beyond the last valid byte of the buffer is determined by “VBState.EndAddress+1”.



2.4 Input Vertex Definition

The 3DSTATE_VERTEX_ELEMENTS command is used to define the source and format of input vertex data and the format of how it is stored in the destination VUE as part of 3DPRIMITIVE processing in the VF unit.

Refer to *3DPRIMITIVE Processing* below for the general flow of how input vertices are input and stored during processing of the 3DPRIMITIVE command.

2.4.1 3DSTATE_VERTEX_ELEMENTS

This is a variable-length command used to specify the active vertex elements (up to 34 [DevSNB+]) Each VERTEX_ELEMENT_STATE structure contains a **Valid** bit which determines which elements are used.

RESTRICTIONS/NOTES:

- At least one VERTEX_ELEMENT_STATE structure must be included.
- **[Pre-DevILK]** Vertex elements must be ordered by increasing **Destination Element Offset**.
- Inclusion of partial VERTEX_ELEMENT_STATE structures is UNDEFINED.
- SW must ensure that at least one vertex element is defined prior to issuing a 3DPRIMTIVE command, or operation is UNDEFINED.
- There are no 'holes' allowed in the destination vertex: NOSTORE components must be overwritten by subsequent components unless they are the trailing DWords of the vertex. Software must explicitly chose some value (probably 0) to be written into DWords that would otherwise be 'holes'.
- Within a VERTEX_ELEMENT_STATE structure, if a Component Control field is set to something other than VFCOMP_STORE_SRC, no higher-numbered Component Control fields may be set to VFCOMP_STORE_SRC. In other words, only trailing components can be set to something other than VFCOMP_STORE_SRC.
- (See additional restrictions listed in the command fields and VERTEX_ELEMENT_STATE description).
- **[DevILK+]** Element[0] must be valid.
- **[DevILK+]** All elements must be valid from Element[0] to the last valid element. (i.e. if Element[2] is valid then Element[1] and Element[0] must also be valid)

[DevILK+] The pitch between elements packed in the URB will always be 128 bits.

DWord	Bit	Description
0	31:29	**Command Type =** GFXPIPE = 03h
	28:16	**GFXPIPE Opcode =** 3DSTATE_VERTEX_ELEMENTS GFXPIPE[28:27 = 3h, 26:24 = 0h, 23:16 = 09h] (Pipelined)
	15:8	Reserved : MBZ
	7:0	**DWord Length** (excludes DWords 0,1) Vertex Element Count = (**DWord Length** + 1) / 2
1-2		**Element[0]** Format: VERTEX_ELEMENT_STATE
[3-4]		**Element[1]**
…		…
[35-36] ... [67-68]		**Element[17]** **[DevSNB+]**... **[DevSNB+] Element[33]**

2.4.2 VERTEX_ELEMENT_STATE Structure

VERTEX_ELEMENT_STATE Structure

Project:	All	Length Bias:	2

This structure is used in 3DSTATE_VERTEX_ELEMENTS to set the state associated with a vertex element. A vertex element is defined as an entity supplying from 1 to 4 DWord vertex components to be stored in the vertex URB entry. Up to 34 (DevSNB+) vertex elements are supported. The VF function will use this state, and possibly the state of the associated vertex buffer, to fetch/generate the source vertex element data, perform any required format conversions, padding with zeros, and store the resulting destination vertex element data into the vertex URB entry.

DWord	Bit	Description
0	31:26	**Vertex Buffer Index** Project: [DevSNB+] Format: U6 FormatDesc Range [0,32] (Up to 33 VBs are supported) This field specifies which vertex buffer the element is sourced from. Programming Notes It is possible for a vertex element to include only internally-generated data (VertexID, etc.), in which case the associated vertex buffer state is ignored.
	25	**Valid** Project: [DevSNB+] Format: Boolean FormatDesc

Value	Name	Description	Project
0h	TRUE	this vertex element is used in vertex assembly	All
1h	FALSE	this vertex element is not used.	All

VERTEX_ELEMENT_STATE Structure

	24:16	**Source Element Format** Project: All Format: The encoding of this field is identical the FormatDesc **Surface Format** field of the SURFACE_STATE structure, as described in the *Sampler* chapter. Range Valid encodings are those marked as "Y" in the "Vertex Buffer" column of the table of Surface Format encodings in the *Sampler* chapter. This field specifies the format in which the memory-resident source data for this particular vertex element is stored in the memory buffer. This only applies to elements stored with VFCOMP_STORE_SRC component control. (All other component types have an explicit format).
	15	**Edge Flag Enable** Project: [DevSNB+] Address: GraphicsAddress[31:0] Surface Type: U32 Range 0..2^32-1 When ENABLED, the source element is interpreted as an EdgeFlag for the vertex. If the source element is zero, the EdgeFlag will be set to FALSE. If the source element is non-zero, the EdgeFlag will be set to TRUE. The EdgeFlag bit will travel down the fixed function pipeline along with the vertex handle, etc. and not be stored in the vertex data like the other vertex elements. Refer to the fixed function descriptions for how this EdgeFlag affects rendering. Edge flags are supported for the following primitive topology types only, otherwise EdgeFlagEnable must not be ENABLED. 3DPRIM_TRILIST* 3DPRIM_TRISTRIP* 3DPRIM_TRIFAN* 3DPRIM_POLYGON **Programming Notes** This bit must only be ENABLED on the last valid VERTEX_ELEMENT structure. When set, Component 0 Control must be set to VFCOMP_STORE_SRC, and Component 1-3 Control must be set to VFCOMP_NOSTORE. **The Source Element Format** must be set to the UINT format. [DevSNB]: Edge Flags are not supported for QUADLIST primitives. Software may elect to convert QUADLIST primitives to some set of corresponding edge-flag-supported primitive types (e.g., POLYGONs) prior to submission to the 3D pipeline.
	14:11	**Reserved** Project: All Format: MBZ

VERTEX_ELEMENT_STATE Structure

	10	**Source Element Offset** (in bytes) Project: All Format: U11 byte offset Range [0,2047 Byte offset of the source vertex element data in the structures comprising the vertex buffer. **Programming Notes** See note on 64-bit float alignment in Buffer Starting Address.
1	31	**Reserved** Project: All Format: MBZ
	30:28	**Component 0 Control** Project: All This field specifies which value is stored for component 0 of this particular vertex element.

Value	Name	Description	Project
0	VFCOMP_NOSTORE	Don't store this component. (Not valid for Component 0, but can be used for Component 1-3). Once this setting is used for a component, all higher-numbered components (if any) MUST also use this setting. (I.e., no holes within any particular vertex element). Also, there are no 'holes' allowed in the destination vertex: NOSTORE components must be overwritten by subsequent components unless they are the trailing DWords of the vertex. Software must explicitly chose some value (probably 0) to be written into DWords that would otherwise be 'holes'.	All

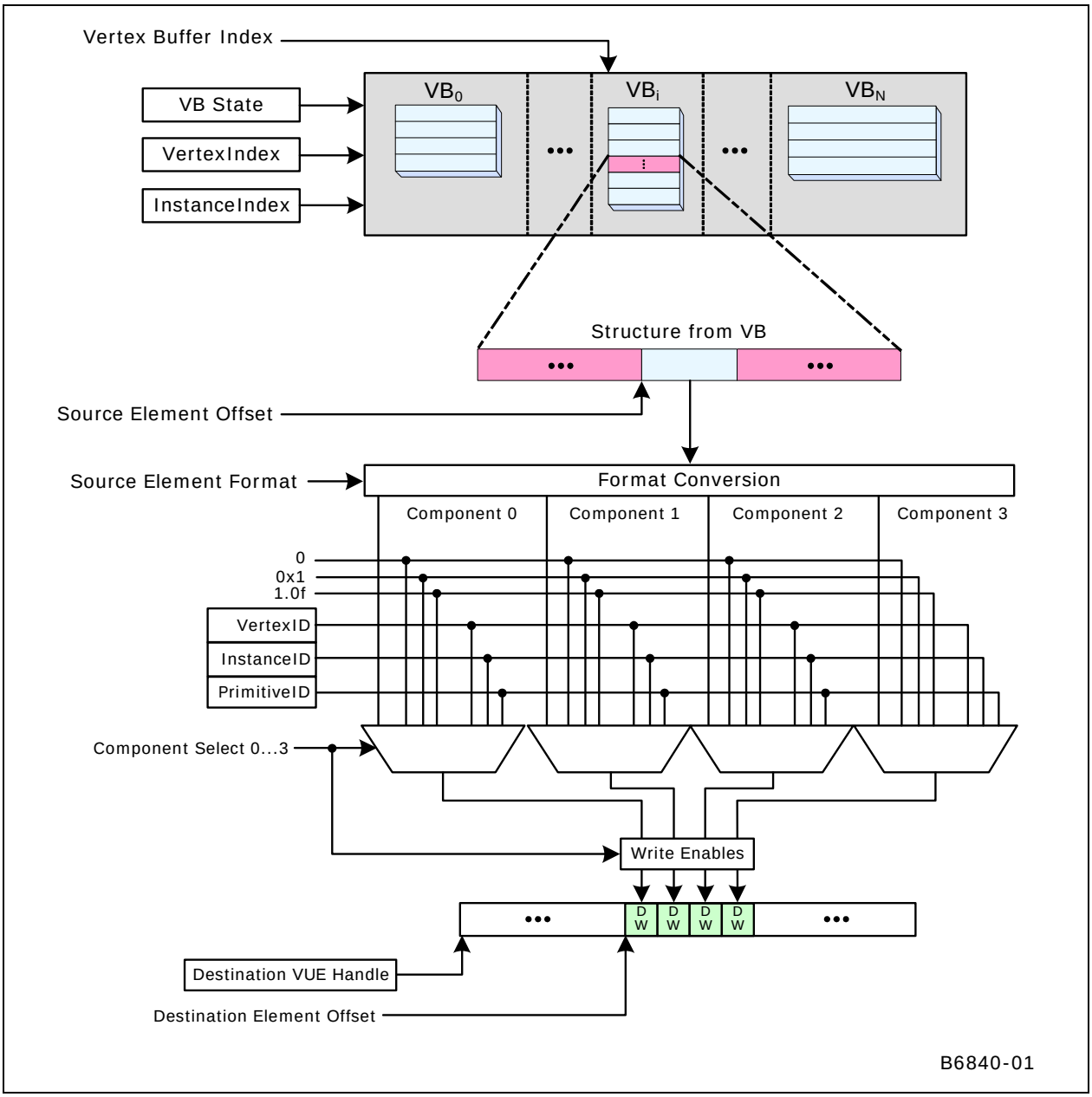
1	VFCOMP_ STORE_SRC	Store corresponding component from format-converted source element. Storing a component that is not included in the Source Element Format results in an UNPREDICTABLE value being stored. Software should used the STORE_0 or STORE_1 encoding to supply default components. Within a VERTEX_ELEMENT_STATE structure, if a Component Control field is set to something other than VFCOMP_STORE_SRC, no higher-numbered Component Control fields may be set to VFCOMP_STORE_SRC. In other words, only trailing components can be set to something other than VFCOMP_STORE_SRC.	All
2	VFCOMP_ STORE_SRC	Store 0 (interpreted as 0.0f if accessed as a float value)	All
3	VFCOMP_ STORE_1_FP	Store 1.0f	All
4	VFCOMP_ STORE_1_ INT	Store 0x1	All
5-6	-	Reserved	
7	VFCOMP_ STORE_PID	Store Primitive ID (as U32) [DevSNB+]: Software should no longer need to use this encoding as PrimitiveID is passed down the FF pipeline – see explanation above).	All

27	**Reserved** Project: All Format: MBZ
26:24	**Component 1 Control**
23	**Reserved** Project: All Format: MBZ
22:20	**Component 2 Control**
19	**Reserved** Project: All Format: MBZ

VERTEX_ELEMENT_STATE Structure		
	18:16	**Component 3 Control**
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**Reserved** Project: Format: MBZ

2.4.3 Vertex Element Data Path

The following diagram shows the path by which a vertex element within the destination VUE is generated and how the fields of the VERTEX_ELEMENT_STATE structure is used to control the generation.



2.5 3D Primitive Processing

2.5.1 3DPRIMITIVE Command [DevSNB]

3DPRIMITIVE			
Project:	DevSNB	**Length Bias:**	2

The 3DPRIMITIVE command is used to submit 3D primitives to be processed by the 3D pipeline. Typically, the processing results in rendering pixel data into the render targets, but this result is not required.

The parameters passed in this command are forwarded to the Vertex Fetch function. The Vertex Fetch function will use this information to generate vertex data structures and store them in the URB. These vertices are then passed down the 3D pipeline for possible processing by the Vertex Shader, Geometry Shader, and Clipper. If rendering is required, the computed vertices are passed down to the StripFan and WindowerMasker units.

DWord	Bit	Description
0	31:29	Command Type Default Value: 3h GFXPIPE Format: OpCode
	28:27	Command SubType Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	3D Command Opcode Default Value: 3h 3DPRIMITIVE Format: OpCode
	23:16	3D Command Sub Opcode Default Value: 0h 3DPRIMITIVE Format: OpCode

3DPRIMITIVE

15 — Vertex Access Type

Project: All

Format: VertexAccessType

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

Value	Name	Description	Project
0h	SEQUENTIAL	VERTEXDATA buffers are accessed sequentially	All
1h	RANDOM	VERTEXDATA buffers are accessed randomly via an index obtained from the Index Buffer.	All

14:10 — Primitive Topology Type

Project: All

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

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

9 — Reserved Project: DevSNB:A Format: MBZ

9 — Indirect Vertex Count Project: DevCTG, DevILK Format: U1

If set, the Vertex Count Per Instance field contains the graphics memory address of the DWord containing the vertex count. If clear, the Vertex Count Per Instance field contains the count as immediate data.

9 — Internal Vertex Count Project: DevSNB:B+ Format: U1

If set, the Vertex Count Per Instance field is ignored and the vertex count is taken from the Internal Vertex Count state register.

8 — Reserved Project: All Format: MBZ

7:0 — DWord Length

Default Value: 4h Excludes DWord (0,1)

Format: =n Total Length - 2

Project: All

3DPRIMITIVE

1	31:0	Vertex Count Per Instance Project: DevBW, DevCL Format: U32 Count of Vertices Range: [0, 2^32-1] upper limit probably constrained by VB size This field specifies how many vertices are to be generated for each instance of the primitive topology. **Programming Notes** This per-instance value should specify a valid number of vertices for the primitive topology type. E.g., for 3DPRIM_TRILIST_ADJ, this field should specify a multiple of 6 vertices. However, in cases where too few or too many vertices are provided, the unused vertices will be silently discarded by the pipeline. A 0 value is this field effectively makes the command a 'no-operation'.
	31:0	Vertex Count Per Instance Project: DevSNB+ Format: U32 Count of Vertices Range: [0, 2^32-1] upper limit probably constrained by VB size This field specifies how many vertices are to be generated for each instance of the primitive topology. Ignored if Internal Vertex Count is set. **Programming Notes** This per-instance value should specify a valid number of vertices for the primitive topology type. E.g., for 3DPRIM_TRILIST_ADJ, this field should specify a multiple of 6 vertices. However, in cases where too few or too many vertices are provided, the unused vertices will be silently discarded by the pipeline. A 0 value is this field effectively makes the command a 'no-operation'.

3DPRIMITIVE		
	31:0	Vertex Count Per Instance Project: DevCTG, DevILK Format: U32 Count of Vertices Range: [0, 2^32-1] upper limit probably constrained by VB size Address: GraphicsAddress[31:0] Surface Type: U32*1 This field specifies how many vertices are to be generated <u>for each instance</u> of the primitive topology. If Indirect Vertex Count is set: Format = DWord-aligned Graphics Memory Address of the count value (there the count value has the same Format/Range as listed below) If Indirect Vertex Count is clear: Format = U32 count of vertices **Programming Notes** This per-instance value should specify a valid number of vertices for the primitive topology type. E.g., for 3DPRIM_TRILIST_ADJ, this field should specify a multiple of 6 vertices. However, in cases where too few or too many vertices are provided, the unused vertices will be silently discarded by the pipeline. A 0 value is this field effectively makes the command a ‘no-operation’.

3DPRIMITIVE

	31:0	BitFieldName Project: All Security: None Access: None Exists If: Always Default Value: 0h DefaultVaueDesc Mask: MMIO(0x2000)#16 Format: U32 FormatDesc Address: GraphicsAddress[31:0] Surface Type: U32 Range 0..2^32-1 BitFieldDesc

Value	Name	Description	Project
0h	Disable	Desc	All
1h	Enable	Desc	All

2	31:0	Start Vertex Location Project: All Format: U32 structure index This field specifies the “starting vertex” for each instance. This allows skipping over part of the vertices in a buffer if, for example, a previous 3DPRIMITIVE command had already drawn the primitives associated with the earlier entries. For SEQUENTIAL access, this field specifies, for each instance, a starting structure index into the vertex buffers For RANDOM access, this field specifies, for each instance, a starting index into the Index Buffer.

Programming Notes
Access of any data outside of the valid extent of a vertex or index buffer will return the value 0 (i.e., appears as if the data stored at the invalid location was 0).

3DPRIMITIVE

3	31:0	Instance Count Project: All Format: U32 count of instances Range 1..2^32-1 This field specifies the number of instances by which the primitive topology is to be regenerated. A value of 0 is UNDEFINED. A value of 1 effectively specifies "non-instanced" operation, though vertex buffers will still be used to provide instance data, if so programmed.
4	31:0	Start Instance Location Project: All Format: U32 structure index This field specifies the "starting instance" for the command as an initial structure index into INSTANCEDATA buffers. Subsequent instances will access sequential instance data structures, as controlled by the **Instance Data Step Rate**. Programming Notes Access of any data outside of the valid extent of a vertex or index buffer will return the value 0 (i.e., appears as if the data stored at the invalid location was 0).
5	31:0	Base Vertex Location Project: All Format: S31 structure index bias This field specifies a signed bias to be added to values read from the index buffer. This allows the same index buffer values to access different vertex data for different commands. This field applies only to RANDOM access mode. This field is ignored for SEQUENTIAL access mode, where there Start Vertex Location can be used to specify different regions in the vertex buffers. Programming Notes Access of any data outside of the valid extent of a vertex or index buffer will return the value 0 (i.e., appears as if the data stored at the invalid location was 0).

The following table defines the encoding of the Primitive Topology Type field. See *3D Pipeline* for details, programming restrictions, diagrams and a discussion of the basic primitive types.

Table 10. 3D Primitive Topology Type Encoding

3D_PrimTopoType

Project: DevSNB

The following table defines the encoding of the Primitive Topology Type field. See 3D Pipeline for details, programming restrictions, diagrams and a discussion of the basic primitive types.

Bit	Description
4:0	Encoding Project: All

Value	Name	Description	Project
00h	Reserved		All
01h	3DPRIM_POINTLIST		All
02h	3DPRIM_LINELIST		All
03h	3DPRIM_LINESTRIP		All
04h	3DPRIM_TRILIST		All
05h	3DPRIM_TRISTRIP		All
06h	3DPRIM_TRIFAN		All
07h	3DPRIM_QUADLIST	The QUADLIST topology is converted to POLYGON topology at the beginning of the 3D pipeline.	All
08h	3DPRIM_QUADSTRIP	The QUADSTRIP topology is converted to POLYGON topology at the beginning of the 3D pipeline.	All
09h	3DPRIM_LINELIST_ADJ		All
0Ah	3DPRIM_LISTSTRIP_ADJ		All
0Bh	3DPRIM_TRILIST_ADJ		All
0Ch	3DPRIM_TRISTRIP_ADJ		All
0Dh	3DPRIM_TRISTRIP_REVERSE		All
0Eh	3DPRIM_POLYGON		All
0Fh	3DPRIM_RECTLIST		All
10h	3DPRIM_LINELOOP	The LINELOOP topology is converted to LINESTRIP topology at the beginning of the 3D pipeline.	All
11h	3DPRIM_POINTLIST _BF		All
12h	3DPRIM_LINESTRIP_CONT		All
13h	3DPRIM_LINESTRIP_BF		All
14h	3DPRIM_LINESTRIP_CONT_BF		All

3D_PrimTopoType				
	15h	Reserved		All
	16h	3DPRIM_TRIFAN_NOSTIPPLE		All
	17h-1Fh	Reserved		All

2.5.2 Functional Overview

The following pseudocode summarizes the general flow of 3D Primitive Processing.

```
CommandInit
InstanceLoop {
    VertexLoop {
        VertexIndexGeneration
        if (CutFlag)
            TerminatePrimitive
        else
            OutputBufferedVertex
            VertexCacheLookup
            if (miss) {
                VertexElementLoop {
                    SourceElementFetch
                    FormatConversion
                    DestinationComponentSelection
                    PrimitiveInfoGeneration
                    URBWrite
                }
            }
    }
    TerminatePrimitive
}
```

2.5.3 CommandInit

The InstanceID value is initialized to 0.

2.5.4 InstanceLoop

The InstanceLoop is the outmost loop, iterating through each instance of primitives. There is no special "non-instanced" mode – at a minimum there is one instance of primitives.

For SEQUENTIAL accessing, the VertexID value is initialized to 0 at the start of each instance. (For RANDOM accessing, there is no initial value for VertexID, as it is derived from the fetched IB value).

The PrimitiveID is also initialized to 0 at the start of each instance. StartPrim is initialized to TRUE.

The VertexLoop (see below) is then executed to iterate through the instance vertices and output vertices to the pipeline as required.

The end of each iteration of InstanceLoop includes an implied "cut" operation.

The InstanceID value is incremented at the end of each InstanceLoop. Note that each instance will produce the same vertex outputs with the exception of any data dependent on InstanceID (i.e., "instance data").

2.5.5 VertexLoop

The VertexLoop iterates VertexNumber through the VertexCountPerInstance vertices for the instance.

For each iteration, a number of processing steps are performed (see below) to generate the information that comprises a vertex. Note that, due to CutProcessing, each iteration does not necessarily output a vertex to the pipeline. When a vertex is to be output, the following information is generated for that vertex:

- PrimitiveType associated with the vertex. This is simply a copy of the PrimitiveTopologyType field of the 3DPRIMITIVE
- VUE handle at which the vertex data is stored
    - For a Vertex Cache hit, the VUE handle is marked with a VCHit boolean, so that the VS unit will not attempt to process (shade) that vertex.
    - Otherwise, the VertexLoop will generate and store the input vertex data into the VUE referenced by this handle.
- The PrimitiveID associated with the vertex. See PrimitiveInfoGeneration.
- PrimStart and PrimEnd booleans associated with the vertex. See PrimitiveInfoGeneration.

(Note that a single vertex of buffering is required in order to associate PrimEnd with a vertex, as this information may not be known until the next iteration through the VertexLoop (see *OutputPrimitiveDelimiter*).

VertexNumber value is incremented by 1 at the end of the loop.

2.5.6 VertexIndexGeneration

A VertexIndex value needs to be derived for each vertex. With the exception of the "cut" index, this index value is used as the vertex cache tag and will be used as a structure index into all VERTEXDATA VBs.

For SEQUENTIAL accessing, the VertexID and VertexIndex value is derived as shown below:

```
VertexIndex = StartVertexLocation + VertexNumber

VertexID = VertexNumber
```

For RANDOM access, the VertexID and VertexIndex is derived from an IBValue read from the IB, as shown below:

```
IBIndex = StartVertexLocation + VertexNumber

VertexID = IB[IBIndex]

if (CutIndexEnable && VertexID == CutIndex)

        CutFlag = 1

else

        VertexIndex = VertexID + BaseVertexLocation

        CutFlag = 0

endif
```

2.5.7 TerminatePrimitive

For RANDOM accessing, and when enabled via **Cut Index Enable**, a fetched IBValue of 'all ones' (0xFF, 0xFFFF, or 0xFFFFFFFF depending on **Index Format**) is interpreted as a 'cut value' and signals the termination of the current primitive and the possible start of the next primitive. This allows the application to specify an instance as a sequence of variable-sized strip primitives (though the cut value applies to any primitive type).

Also, there is an implied primitive termination at the end of each InstanceLoop (and so strip primitives cannot span multiple instances).

In either case, the currently-buffered vertex (if any) is marked with EndPrim and then flushed out to the pipeline.

The next-output vertex (if any) will be marked with StartPrim.

Whenever a primitive delimiter is encountered, the PIDCounterS and PIDCounterR counters are reset to 0. These counters control the incrementing (in PrimitiveInfoGeneration, below) of PrimitiveID within each primitive topology of an instance.

```
if (PIDCounterS != 0) // There is a buffered vertex

        if (primType == TRISTRIP_ADJ)
```

```
                if (PIDCounterS==6 || PIDCounterR==1)
                        PrimitiveID++
                endif
        endif
        PrimEnd = TRUE
        OutputBufferedVertex
endif
PrimEnd = FALSE
PrimStart = TRUE
```

2.5.8 VertexCacheLookup

The VertexIndex value is used as the tag value for the VertexCache (see *Vertex Cache*, above). If the Vertex Cache is enabled and the VertexIndex value hits in the cache, the VUE handle is read from the cache and inserted into the vertex stream. It is marked with a VCHit boolean to surpress processing (shading) in the VS unit.

Otherwise, for Vertex Cache misses, a VUE handle is obtained to provide storage for the generated vertex data. VertexLoop processing then proceeds to iterate through the VEs to generate the destination VUE data.

2.5.9 VertexElementLoop

The VertexElementLoop generates and stores vertex data in the destination VUE one VE at a time.

[Pre-DevILK] Note that VEs must be defined (via 3DSTATE_VERTEX_ELEMENTS) in order of increasing **Destination Element Offset**, though architecturally the order by which VEs are processed is arbitrary (has no impact on the results).

2.5.10 SourceElementFetch

The following assumes the VE requires data from a VB, which is the typical case. In the case that the VE is completely comprised of constant and/or auto-generated IDs, the SourceElementFetch and FormatConversion steps are skipped.

The structure index within the VE's selected VB is computed as follows:

```
if (VB is a VERTEXDATA VB)
        VBIndex = VertexIndex
else // INSTANCEDATA VB
```

```
        VBIndex = StartInstanceLocation

        if (VB.InstanceDataStepRate > 0)

                VBIndex += InstanceID/VB.InstanceDataStepRate

    endif
```

If VBIndex is invalid (i.e., negative or past **Max Index**), the data returned from the VB fetch is defined to be zero. Otherwise, the address of the source data required for the VE is then computed and the data is read from the VB. The amount of data read from the VB is determined by the **Source Element Format**.

```
    if ( (VBIndex<0) || (VBIndex>VB.MaxIndex) )

        srcData = 0

    else

        pSrcData = VB.BufferStartingAddress + (VBIndex * VB.BufferPitch) +
    VE.SourceElementOffset

        srcData = MemoryRead( pSrcData, VE.SourceElementFormat )

    endif
```

2.5.11 FormatConversion

Once the VE source data has been fetched, it is subjected to format conversion. The output of format conversion is up to 4 32-bit components, each either integer or floating-point (as specified by the **Source Element Format**). See *Sampler* for conversion algorithms.

The following table lists the valid **Source Element Format** selections, along with the format and availability of the converted components (if a component is listed as "-", it cannot be used as source of a VUE component). Note: This table is a subset of the list of supported surface formats defined in the *Sampler* chapter. Please refer to that table as the "master list". This table is here only to identify the components available (per format) and their format.

Table 11. Source Element Formats supported in VF Unit

Source Element Format	Converted Component				
	Format	0	1	2	3
256 bits					
R64G64B64A64_FLOAT	FLOAT	R	G	B	A
192 bits					
R64G64B64_FLOAT	FLOAT	R	G	B	A
128 bits					

Source Element Format	Converted Component				
	Format	0	1	2	3
R32G32B32A32_FLOAT	FLOAT	R	G	B	A
R32G32B32A32_SNORM	FLOAT	R	G	B	A
R32G32B32A32_UNORM	FLOAT	R	G	B	A
R32G32B32A32_SINT	SINT	R	G	B	A
R32G32B32A32_UINT	UINT	R	G	B	A
R32G32B32A32_SSCALED	FLOAT	R	G	B	A
R32G32B32A32_USCALED	FLOAT	R	G	B	A
R32G32B32A32_SFIXED	FLOAT	R	G	B	A
R64G64_FLOAT	FLOAT	R	G	-	-
96 bits					
R32G32B32_FLOAT	FLOAT	R	G	B	-
R32G32B32_SNORM	FLOAT	R	G	B	-
R32G32B32_UNORM	FLOAT	R	G	B	-
R32G32B32_SINT	SINT	R	G	B	-
R32G32B32_UINT	UINT	R	G	B	-
R32G32B32_SSCALED	FLOAT	R	G	B	-
R32G32B32_USCALED	FLOAT	R	G	B	-
R32G32B32_SFIXED	FLOAT	R	G	B	-
64 bits					
R16G16B16A16_FLOAT	FLOAT	R	G	B	A
R16G16B16A16_SNORM	FLOAT	R	G	B	A
R16G16B16A16_UNORM	FLOAT	R	G	B	A
R16G16B16A16_SINT	SINT	R	G	B	A
R16G16B16A16_UINT	UINT	R	G	B	A
R16G16B16A16_SSCALED	FLOAT	R	G	B	A
R16G16B16A16_USCALED	FLOAT	R	G	B	A

Source Element Format	Converted Component				
	Format	0	1	2	3
R32G32_FLOAT	FLOAT	R	G	-	-
R32G32_SNORM	FLOAT	R	G	-	-
R32G32_UNORM	FLOAT	R	G	-	-
R32G32_SINT	SINT	R	G	-	-
R32G32_UINT	UINT	R	G	-	-
R32G32_SSCALED	FLOAT	R	G	-	-
R32G32_USCALED	FLOAT	R	G	-	-
R32G32_SFIXED	FLOAT	R	G	-	-
R64_FLOAT	FLOAT	R	-	-	-
48 bits					
R16G16B16_FLOAT [**DevSNB+**]	FLOAT	R	G	B	-
R16G16B16_SNORM	FLOAT	R	G	B	-
R16G16B16_UNORM	FLOAT	R	G	B	-
R16G16B16_UINT	UINT	R	G	B	-
R16G16B16_SINT	SINT	R	G	B	-
R16G16B16_SSCALED	FLOAT	R	G	B	-
R16G16B16_USCALED	FLOAT	R	G	B	-
32 bits					
R10G10B10A2_UNORM	FLOAT	R	G	B	A
R10G10B10A2_SNORM	FLOAT	R	G	B	A
R10G10B10A2_USCALED	FLOAT	R	G	B	A
R10G10B10A2_SSCALED	FLOAT	R	G	B	A
B10G10R10A2_UNORM	FLOAT	R	G	B	A
B10G10R10A2_SNORM	FLOAT	R	G	B	A
B10G10R10A2_USCALED	FLOAT	R	G	B	A
B10G10R10A2_SSCALED	FLOAT	R	G	B	A

Source Element Format	Converted Component				
	Format	0	1	2	3
R10G10B10A2_UINT	UINT	R	G	B	A
R10G10B10X2_USCALED	FLOAT	R	G	B	-
R10G10B10_SNORM_A2_UNORM	FLOAT	R	G	B	A
R8G8R8A8_UNORM	FLOAT	B	G	R	A
R8G8B8A8_SNORM	FLOAT	R	G	B	A
R8G8B8A8_UNORM	FLOAT	R	G	B	A
R8G8B8A8_SINT	SINT	R	G	B	A
R8G8B8A8_UINT	UINT	R	G	B	A
R8G8B8A8_SSCALED	FLOAT	R	G	B	A
R8G8B8A8_USCALED	FLOAT	R	G	B	A
R11G11B10_FLOAT	FLOAT	R	G	B	-
R16G16_FLOAT	FLOAT	R	G	-	-
R16G16_SNORM	FLOAT	R	G	-	-
R16G16_UNORM	FLOAT	R	G	-	-
R16G16_SINT	SINT	R	G	-	-
R16G16_UINT	UINT	R	G	-	-
R16G16_SSCALED	FLOAT	R	G	-	-
R16G16_USCALED	FLOAT	R	G	-	-
R32_FLOAT	FLOAT	R	-	-	-
R32_SINT	SINT	R	-	-	-
R32_UINT	UINT	R	-	-	-
R32_SSCALED	FLOAT	R	-	-	-
R32_USCALED	FLOAT	R	-	-	-
R32_SNORM	FLOAT	R	-	-	-
R32_UNORM	FLOAT	R	-	-	-

Source Element Format	Converted Component				
	Format	0	1	2	3
R32_SFIXED	FLOAT	R	-	-	-
24 bits					
R8G8B8_SNORM	FLOAT	R	G	B	-
R8G8B8_UNORM	FLOAT	R	G	B	-
R8G8B8_SSCALED	FLOAT	R	G	B	-
R8G8B8_USCALED	FLOAT	R	G	B	-
16 bits					
R8G8_SNORM	FLOAT	R	G	-	-
R8G8_UNORM	FLOAT	R	G	-	-
R8G8_SINT	SINT	R	G	-	-
R8G8_UINT	UINT	R	G	-	-
R8G8_SSCALED	FLOAT	R	G	-	-
R8G8_USCALED	FLOAT	R	G	-	-
R16_FLOAT	FLOAT	R	-	-	-
R16_SNORM	FLOAT	R	-	-	-
R16_UNORM	FLOAT	R	-	-	-
R16_SINT	SINT	R	-	-	-
R16_UINT	UINT	R	-	-	-
R16_SSCALED	FLOAT	R	-	-	-
R16_USCALED	FLOAT	R	-	-	-
8 bits					
R8_SNORM	FLOAT	R	-	-	-
R8_UNORM	FLOAT	R	-	-	-
R8_SINT	SINT	R	-	-	-
R8_UINT	UINT	R	-	-	-
R8_SSCALED	FLOAT	R	-	-	-

Source Element Format	Converted Component				
	Format	0	1	2	3
R8_USCALED	FLOAT	R	-	-	-

2.5.12 DestinationFormatSelection

The **Component Select 0..3** bits are then used to select, on a per-component basis, which destination components will be written and with which value. The supported selections are the converted source component, VertexID, InstanceID, PrimitiveID, the constants 0 or 1.0f, or nothing (VFCOMP_NO_STORE). If a converted component is listed as '-' (not available) in Table 11, it must not be selected (via VFCOMP_STORE_SRC), or an UNPREDICTABLE value will be stored in the destination component.

The selection process sequences from component 0 to 3. Once a **Component Select** of VFCOMP_NO_STORE is encountered, all higher-numbered **Component Select** settings must also be programmed as VFCOMP_NO_STORE. It is therefore not permitted to have 'holes' in the destination VE.

2.5.13 PrimitiveInfoGeneration

A PrimitiveID value and PrimStart boolean need to be associated with the vertex.

If the vertex is either the first vertex of an instance or the first vertex following a 'cut index', the vertex is marked with PrimStart.

PrimitiveID gets incremented such that subsequent per-object processing (i.e., in the GS or SF/WM) will see an incrementing value associated with each sequential object within an instance. The PrimitiveID associated with the provoking, non-adjacent vertex of an object is applied to the object.

The following pseudocode describe the logic used in the VertexLoop to compute the PrimitiveID value associated with the vertex. Recall that PrimitiveID is reset to 0 at the start of each InstanceLoop.

```
if (PIDCounterS < S[primType])
        PIDCounterS++
else
        if (PIDCounterR < R[primType])
                PIDCounterR++
        else
                PrimitiveID++
                PIDCounterR = 0
        endif
endif
```

Two counters are employed to control the incrementing of PrimitiveID. The counters are compared against two corresponding parameters associated with the primitive topology type.

The PIDCounterS is used to 'skip over' some number (possibly zero) initial vertices of the primitive topology. This counter gets reset to 0 after each primitive is terminated.

Then the PIDCounterR is used to periodically increment the PrimitiveID, where the incrementing interval (vertex count) is topology-specific.

The following table lists the S[] and R[] values associated with each primitive topology type.

PrimTopologyType	S, R	PrimitiveID Outputs
POINTLIST POINTLIST_BF	1, 0	0,1,2,3, …
LINELIST	1, 1	0,0,1,1,2,2,3,3, …
LINELIST_ADJ	1, 3	0,0,0,0,1,1,1,1,2,2,2,2,3,3,3,3 …
LINESTRIP LINESTRIP_BF LINESTRIP_CONT	2, 0	0,0,1,2,3, …
LINESTRIP_ADJ	3, 0	0,0,0,1,2,3,…
LINELOOP	2, 0	0,0,1,2,3,… Note: this breaks the usage model (as the initial vertex is the provoking vertex for the closing line, but it has an invalid PrimitiveID of 0), but is effectively a don't care as PrimitiveID is only required for D3D and LINELOOP is an OpenGL-only primitive.) The LINELOOP topology is converted to LINESTRIP topology at the beginning of the 3D pipeline.
TRILIST RECTLIST	1, 2	0,0,0,1,1,1,2,2,2,3,3,3,…
TRILIST_ADJ	1, 5	0,0,0,0,0,0,1,1,1,1,1,1,2,2,2,2,2,2,…
TRISTRIP TRISTRIP_REV	3, 0	0,0,0,1,2,3, …
TRISTRIP_ADJ	5, 1	0,0,0,0,0,0,1,1,2,2,3,3, …
TRIFAN TRIFAN_NOSTIPPLE POLYGON	3, 0	0,0,0,1,2,3, …

PrimTopologyType	S, R	PrimitiveID Outputs
QUADLIST	1, 3	0,0,0,0,1,1,1,1,2,2,2,2,3,3,3,3, … Note: The QUADLIST topology is converted to POLYGON topology at the beginning of the 3D pipeline.
QUADSTRIP	3, 1	0,0,0,0,1,1,2,2,3,3, … Note: The QUADSTRIP topology is converted to POLYGON topology at the beginning of the 3D pipeline.
Reserved		

2.5.14 URBWrite

The selected destination components are written into the destination VUE starting at **Destination Offset Select**. See the description of 3DPRIMITIVE for restrictions on this field.

2.5.15 OutputBufferedVertex

In order to accommodate 'cut' processing, the VF unit buffers one output vertex. The generation of a new vertex or the termination of a primitive causes the buffered vertex to be output to the pipeline.

2.6 Dangling Vertex Removal

The last functional stage of processing of the 3DPRIMITIVE command is the removal of "dangling" vertices. This includes the discarding of primitive topologies without enough vertices for a single object (e.g., a TRISTRIP with only two vertices), as well as the discarding of trailing vertices that do not form a complete primitive (e.g., the last two vertices of a 5-vertex TRILIST).

This function is best described as a filter operating on the vertex stream emitted from the processing of the 3DPRIMITIVE. The filter inputs the PrimType, PrimStart and PrimEnd values associated with the generated vertices. The filter only outputs primitive topologies without dangling vertices. This requires the filter to (a) be able to buffer some number of vertices, and (b) be able to remove dangling vertices from the pipeline and dereference the associated VUE handles.

2.7 Other Vertex Fetch Functionality

2.7.1 Statistics Gathering

The VF stage tracks two pipeline statistics, the number of vertices fetched and the number of objects generated. VF will increment the appropriate counter for each when statistics gathering is enabled by issuing the 3DSTATE_VF_STATISTICS command with the **Statistics Enable** bit set.

DWord	Bit	Description
0	31:29	**Command Type =** GFXPIPE = 03h
	28:16	**GFXPIPE Opcode =** 3DSTATE_VF_STATISTICS **[DevBW], [DevCL]** GFXPIPE[28:27 = 3h, 26:24 = 0h, 23:16 = 0Bh] (Pipelined) **[DevCTG+]** GFXPIPE[28:27 = 1h, 26:24 = 0h, 23:16 = 0Bh] (Pipelined, Single DW)
	15:1	Reserved : MBZ
	0	**Statistics Enable** If ENABLED, VF will increment the pipeline statistics counters IA_VERTICES_COUNT and IA_PRIMITIVES_COUNT for each vertex fetched and each object output, respectively, for 3DPRIMITIVE commands issued subsequently. If DISABLED, these counters will not be incremented for subsequent 3DPRIMITIVE commands. Format: Enable

2.7.1.1 Vertices Generated

VF will increment the IA_VERTICES_COUNT Register (see Memory Interface Registers in Volume Ia, *GPU*) for each vertex it fetches, even if that vertex comes from a cache rather than directly from a vertex buffer in memory. Any “dangling” vertices (fetched vertices that are part of an incomplete object) should be included.

2.7.1.2 Objects Generated

VF will increment the IA_PRIMITIVES_COUNT Register (see Memory Interface Registers in Volume Ia, *GPU*) for each object (point, line, triangle or quadrilateral) that it forwards down the pipeline. NOTE: For LINELOOP, the last (closing) line object is not counted.

3. Vertex Shader (VS) Stage

3.1 VS Stage Overview

The VS stage of the 3D Pipeline is used to perform processing ("shading") of vertices after being assembled and written to the URB by the VF function. The primary function of the VS stage is to pass vertices that miss in the Vertex Cache to VS threads, and then pass the VS thread-generated vertices down the pipeline. Vertices that hit in the Vertex Cache are passed down the pipeline unmodified.

When the VS stage is disabled, vertices flow through the unit unmodified (i.e., as written by the VF unit).

Refer to the *Common 3D FF Unit Functions* subsection in the *3D Overview* chapter for a general description of a 3D pipeline stage, as much of the VS stage operation and control falls under these "common" functions. I.e., most stage state variables and VS thread payload parameters are described in *3D Overview*, and although they are listed here for completeness, that chapter provides the detailed description of the associated functions.

Refer to this chapter for an overall description of the VS stage, and any exceptions the VS stage exhibits with respect to common FF unit functions.

3.1.1 Vertex Caching

The 3D Pipeline employs a Vertex Cache that is shared between the VF and VS units. (See *Vertex Fetch* chapter for additional information). The Vertex Cache may be explicitly DISABLED via the **Vertex Cache Disable** bit in VS_STATE. Even when explicitly ENABLED, the VS unit can (by default) <u>implicitly</u> disable and invalidate the Vertex Cache when it detects one of the following conditions:

1. Either VertexID or PrimitiveID is selected as part of the vertex data stored in the URB.

2. Sequential indices are used in the 3DPRIMITIVE command (though this is effectively a don't care as there wouldn't be any hits anyway).

The implicit disable will persist as long as one of these conditions persist. The **Vertex Cache Implicit Disable Inhibit** bit in the VFSKPD MI register is provided to inhibit the VS unit's implicit cache disable. If inhibited, software is responsible for explicitly enabling/disabling the vertex cache as required for correct operation.

Note: Even though use of VertexID causes an implicit cache disable, there is no known (good) reason why this is required. Software can therefore allow the implicit cache disable (the default action) and live with some possible performance penalty due to the too-often-disabled cache.

The Vertex Cache is implicitly invalidated between 3DPRIMITIVE commands and between instances within a 3DPRIMITIVE command – therefore use of InstanceID in a Vertex Element is not a condition underwhich the cache is implicitly disabled.

The following table summarizes the modes of operation of the Vertex Cache:

Vertex Cache	VS Function Enable	Mode of Operation
DISABLED (implicitly or explicitly)	DISABLED	Vertex Cache is not used. VF unit will assemble all vertices and write them into the URB entry supplied by the VS unit. VS unit will pass references to these VUEs down the pipeline unmodified. Usage Model: This is an exceptional condition, only required when the VF-generated vertices contain InstanceID or PrimitiveID and more than one instance is produced. Otherwise the Vertex Cache should be enabled.
DISABLED (implicitly or explicitly)	ENABLED	Vertex Cache is not used. VF unit will assemble all vertices and write them into the URB entry supplied by the VS unit. VS unit will spawn VS threads to process all vertices, overwriting the input data with the results. The VS unit pass references to these VUEs down the pipeline. Usage Model: This mode is only used when the VS function is required, but either (a) the input vertex contains InstanceID or PrimitiveID and more than one instance is generated or (b) the VS kernel produces a side effect (e.g., writes to a memory buffer) which requires every vertex to be processed by a VS thread.
ENABLED	DISABLED	Vertex Cache is used to provide reuse of VF-generated vertices. The VF unit will check the cache and only process (assemble/write) vertices that miss in the cache. In either case, the VS unit will pass references to vertices (that hit or miss) down the pipeline without spawning any VS threads. Usage Model: Normal operation when the VS function is <u>not</u> required. Note that there may be situations which require the VS function to be used even when not explicitly required by the API. E.g., perspective divide may be required for clip testing.
ENABLED	ENABLED	Vertex Cache is used to provide reuse of VS-processed vertices. The VF unit will check the cache and only process (assemble/write) vertices that miss in the cache. The VS unit will only process (shade) the vertices that missed in the cache. The VS unit sends references to hit or missed vertices down the pipeline in the correct order. Usage Model: Normal operation when the VS function is required and use of the Vertex Cache is permissible.

3.2 VS Stage Input

As a stage of the 3D pipeline, the VS stage receives inputs from the previous (VF) stage. Refer to *3D Overview* for an overview of the various types of input to a 3D Pipeline stage. The remainder of this subsection describes the inputs specific to the VS stage.

3.2.1 State

3.2.1.1 URB_FENCE

Refer to *3D Overview* for a description of how the VS stage processes this command.

3.2.1.2 VS_STATE [Pre-DevSNB]

The following table describes the format and contents of the VS_STATE structure referenced by the **Pointer to VS State** field of the 3DSTATE_PIPELINED_POINTERS command.

<table>
<tr><th colspan="4">3DSTATE_VS</th></tr>
<tr><td colspan="2">Project: [Pre-DevSNB]</td><td>Length Bias:</td><td>2</td></tr>
<tr><td colspan="4">For [Pre-DevSNB], the state used by VS is defined with this inline state packet.</td></tr>
<tr><th>DWord</th><th>Bit</th><th colspan="2">Description</th></tr>
<tr><td rowspan="2">0</td><td>31:6</td><td colspan="2">Kernel Start Pointer
Project: All
Format: [Pre-DevIL]: Format = GeneralStateOffset[31:6] FormatDesc
[DevIL]: Format = InstructionBaseOffset[31:6]
Address: GraphicsAddress[31:0]
Surface Type: U32
Range 0..2^32-1
This field specifies the starting location (1st core instruction) of the kernel program run by threads spawned by this FF unit. It is specified as a 64-byte-granular offset from the General State Base Address [Pre-DevIL] or Instruction Base Address [DevIL].
This field is ignored if VS Function Enable is DISABLED.
<table>
<tr><th>Errata</th><th>Description</th><th>Project</th></tr>
<tr><td>[BWT007</td><td>Instructions pointed at by offsets from General State must be contained within 32-bit physical address space (that is, must map to memory pages under 4G.)</td><td>[DevBW-A,B]</td></tr>
</table></td></tr>
<tr><td>5:4</td><td colspan="2">Reserved Project: All Format: MBZ</td></tr>
</table>

	3:1	**GRF Register Count** Project: All Security: None Access: None Exists If: Always Default Value: 0h DefaultVaueDesc Mask: MMIO(0x2000)#16 Format: U32 FormatDesc Address: GraphicsAddress[31:0] Surface Type: U32 Range 0..2^32-1 Defines the number of GRF Register Blocks used by the kernel. A register block contains 16 registers. A kernel using a register count that is not a multiple of 16 must round up to the next multiple of 16. This field is ignored if **VS Function Enable** is DISABLED.
	0	**Reserved** Project: All Format: MBZ
1	31	**Single Program Flow (SPF)** Specifies whether the kernel program has a single program flow (SIMDnxm with m = 1) or multiple program flows (SIMDnxm with m > 1). If set, the VS unit will only dispatch 1-vertex thread payloads. See CR0 description in *ISA Execution Environment*.
	30:26	**Reserved** Project: All Format: MBZ

Value	Name	Description	Project
0	Multiple Program Flows	Multiple Program Flows (1- or 2-vertex threads spawned, operating under normal (SIMD4x2) mode)	All
1	Single Program Flow	Single Program Flow (only 1-vertex threads spawned, operating under SPF EU mode)	All

Programming Notes
This field is ignored if **VS Function Enable** is DISABLED.

3DSTATE_VS

25:18 Binding Table Entry Coun

Project: All

Format: U8 FormatDesc

Range [0,255]

Specifies whether the kernel program has a single program flow (SIMDnxm with m = 1) or multiple program flows (SIMDnxm with m > 1). If set, the VS unit will only dispatch 1-vertex thread payloads. See CR0 description in *ISA Execution Environment*.

Programming Notes

This field is ignored if **VS Function Enable** is DISABLED.

[DevILK:A], [DevILK:B] MBZ

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

17 Thread Priority

Project: All

Specifies the priority of the thread for dispatch:

Value	Name	Description	Project
0	Normal	Normal priority	All
1	High	High priority	All

Programming Notes

This field is ignored if **VS Function Enable** is DISABLED.

[Pre-DevIL]: this field must be zero.

16 Floating Point Mode:

Project: All

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

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

Programming Notes

This field is ignored if **VS Function Enable** is DISABLED.

3DSTATE_VS

15:14	Reserved Project: All Format: MBZ
13	Illegal Opcode Exception Enable Project: All Format Enable FormatDesc This bit gets loaded into EU CR0.1[12] (note the bit # difference). See *Exceptions* and *ISA Execution* Environment. **Programming Notes** This field is ignored if **VS Function Enable** is DISABLED.
12	Reserved Project: All Format: MBZ
11	MaskStack Exception Enable Project: All Format: Enable FormatDesc This bit gets loaded into EU CR0.1[12] (note the bit # difference). See *Exceptions* and *ISA Execution Environment*. **Programming Notes** This field is ignored if **VS Function Enable** is DISABLED.
10:8	Reserved Project: All Format: MBZ
7	Software Exception Enable Project: All Format Enable FormatDesc This bit gets loaded into EU CR0.1[13] (note the bit # difference). See *Exceptions* and *ISA Execution Environment*. **Programming Notes** This field is ignored if **VS Function Enable** is DISABLED.
6:0	Reserved Project: All Format: MBZ

3DSTATE_VS		
2	31:10	Scratch Space Base Of Project: All Format: GeneralStateOffset[31:10] FormatDesc Range 0..2^32-1 Specifies the starting location of the scratch space area allocated to this FF unit as a 1K-byte aligned offset from the **General State Base Address**. If required, each thread spawned by this FF unit will be allocated some portion of this space, as specified by **Per-Thread Scratch Space**. The computed offset of the thread-specific portion will be passed in the thread payload as **Scratch Space Offset**. The thread is expected to utilize "stateless" DataPort read/write requests to access scratch space, where the DataPort will cause the **General State Base Address** to be added to the offset passed in the request header. **Programming Notes** This field is ignored if **VS Function Enable** is DISABLED.
	9:4	Reserved Project: All Format: MBZ
	3:0	Per-Thread Scratch Space Project: All Format: U4 power of 2 Bytes over 1K Bytes FormatDesc Range [0,11] indicating [1K Bytes, 2M Bytes] Specifies the amount of scratch space to be allocated to each thread spawned by this FF unit. The driver must allocate enough contiguous scratch space, starting at the **Scratch Space Base Pointer**, to ensure that the **Maximum Number of Threads** can each get **Per-Thread Scratch Space** size without exceeding the driver-allocated scratch space. **Programming Notes** This field is ignored if **VS Function Enable** is DISABLED. This amount is available to the kernel for information only. It will be passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port will ignore it.
3	31	Reserved Project: All Format: MBZ
	30:25	**Constant URB Entry Read Length** Project: All Format: U6 FormatDesc Range [0,63] Specifies the amount of URB data read and passed in the thread payload <u>for the Constant URB entry</u>, in 256-bit register increments. **Programming Notes** This field is ignored if **VS Function Enable** is DISABLED.

3DSTATE_VS		
	24	Reserved Project: All Format: MBZ
	23:18	Constant URB Entry Read Offset Project: All Format: U6 FormatDesc Range [0,63] Specifies the amount of URB data read and passed in the thread payload for the Constant URB entry, in 256-bit register increments. Programming Notes This field is ignored if **VS Function Enable** is DISABLED.
	17	Reserved Project: All Format: MBZ
	16:11	Vertex URB Entry Read Length Project: All Format: U6 FormatDesc Range [1,63] Specifies the amount of URB data read and passed in the thread payload for each Vertex URB entry, in 256-bit register increments. Programming Notes This field is ignored if **VS Function Enable** is DISABLED. It is UNDEFINED to set this field to 0 indicating no Vertex URB data to be read and passed to the thread.
	10	Reserved Project: All Format: MBZ
	9:4	Vertex URB Entry Read Offset Project: All Format: U6 FormatDesc Range [0,63] Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB before being included in the thread payload. This offset applies to all Vertex URB entries passed to the thread. Programming Notes This field is ignored if **VS Function Enable** is DISABLED.

3DSTATE_VS		
	3:0	Dispatch GRF Start Register for URB Data Project: All Format: U4 FormatDesc Range [0,15] indicating GRF [R0,R15] Specifies the starting GRF register number for the URB portion (Constant + Vertices) of the thread payload. Programming Notes This field is ignored if **VS Function Enable** is DISABLED.
4	31	Reserved Project: All Format: MBZ
	30:25	Maximum Number of Threads Project: All Format: U5 representing thread count - 1 FormatDesc Range [Pre-DevCTG:B]Range = [0,15] indicating thread count of [1,16] [DevCTG:B+]Range = [0,31] indicating thread count of [1,32] [DevILK]Range = [0, 71] indicating thread count of [1,72] Specifies the maximum number of simultaneous threads allowed to be active. Used to avoid using up the scratch space, or to avoid potential deadlock. Programming Notes This field is ignored if **VS Function Enable** is DISABLED.
	24	Reserved Project: All Format: MBZ
	23:19	URB Entry Allocation Size Project: All Format: U5 count (of 512-bit units) – 1 FormatDesc Range [0,31] = [1,32] 512-bit units = [2,64] 256-bit URB rows Specifies the length of each URB entry owned by this FF unit. Programming Notes This field is always used (even if VS Function Enable is DISABLED). Changing this value requires a subsequent URB_FENCE command. See Graphics Processing Engine for Command Ordering Rules and a description of URB_FENCE.
	18	Reserved Project: All Format: MBZ
	17:11	Number of URB Entries Project: All Format: **[DevILK]** FormatDesc Format = U9 shift right by 2, see valid settings below

		Range **[DevCTG-B]**: Format = U7, see valid settings below **[Pre DevCTG-B]**: Format = U6, see valid settings below **DevBW-A,B Restriction**: Format = U6, see valid settings below **[DevILK]** Range = [2=8 entries, 3=12 entries, 4=16 entries, 8=32 entries, 16=64 entries, 24=96 entries, 32=128 entries, 42=168 entries, 48=192 entires, 56=224 entries, 64=256 entries] (see restriction above) **[DevCTG-B]**: Range = [8,12, 16, 32, 64] (see restriction above) **[Pre DevCTG-B]**: Range = [8,12, 16, 32] (see restriction above) **DevBW-A,B Restriction**: Range = [8,12, 16] Specifies the number of URB entries that are used by this FF unit. Programming Notes This field is always used (even if **VS Function Enable** is DISABLED). Changing this value requires a subsequent URB_FENCE command. See Graphics Processing Engine for Command Ordering Rules and a description of URB_FENCE. This field must be programmed to 12 or greater in order to process TRISTRIP_ADJ primitives, otherwise operation is UNDEFINED (possible hang).
	10	Statistics Enable Project: All Format: Enable FormatDesc Address: GraphicsAddress[31:0] Surface Type: U32 Range 0..2^32-1 If ENABLED, this FF unit will engage in statistics gathering. See the *Statistics Gathering* section later in this chapter. If DISABLED, statistics information associated with this FF stage will be left unchanged. Programming Notes This field is effectively if **VS Function Enable** is DISABLED.
	9:0	Reserved Project: All Format: MBZ

3DSTATE_VS

5 — 31:5

Sampler State Offset

Project: All

Format: GeneralStateOffset[31:5] FormatDesc

This field, together with the **General State Base Address**, specifies the starting location of the Sampler State Table used by threads spawned by this FF unit. It is specified as a 32-byte-granular offset from the **General State Base Address**. The Sampler will apply the offset to the **General State Base Address** when accessing Sampler State data.

Programming Notes
This field is ignored if **VS Function Enable** is DISABLED.

Errata	Description	Project
Errata BWT007	Sampler state pointed at by offsets from General State must be contained within 32-bit physical address space (that is, must map to memory pages under 4G.)	[DevBW-A,B]

5 — 4:3

Reserved Project: All Format: MBZ

5 — 2:0

Sampler Count

Project: All but ILK

Format: U3 FormatDesc

Range [0,4]

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

0: no samplers used

1: between 1 and 4 samplers used

2: between 5 and 8 samplers used

3: between 9 and 12 samplers used

4: between 13 and 16 samplers used

Programming Notes
This field is ignored if **VS Function Enable** is DISABLED.

5 — 31:0

Reserved Project: [DevILK:A-B] Format: MBZ

6 — 31:2

Reserved Project: All Format: MBZ

1

Vertex Cache Di

Project: All

Format: Disable FormatDesc

This bit controls the operation of the Vertex Cache. This field is always used.

Value	Name	Description	Project
0	Vertex Cache is DISABLED and the VS Function is ENABLED	the Vertex Cache is not used and all incoming vertices will be passed to VS threads.	All
1	Vertex Cache is ENABLED and the VS Function is ENABLED	incoming vertices that do not hit in the Vertex Cache will be passed to VS threads.	All
2	Vertex Cache is ENABLED and the VS Function is DISABLED	input vertices that miss in the Vertex Cache will be assembled and written to the URB, though pass thru the VS stage unmodified (not shaded).	

Programming Notes
The Vertex Cache is invalidated whenever the Vertex Cache becomes DISABLED , whenever the VS Function Enable toggles, between 3DPRIMITIVE commands and between instances within a 3DPRIMITIVE command. See the Vertex Caching section (above) for details on implicit vertex cache disabling.

0

VS Function Enable

Project: All

Format: Enable FormatDesc

This field is always used.

Value	Name	Description	Project
0	Disable	VF-generated vertices will pass thru the VS function and sent down the pipeline unmodified. The Vertex Cache is still available in this mode, if enabled.	All
1	Enable	VF-generated vertices will pass thru the VS function and sent down the pipeline unmodified. The Vertex Cache is still available in this mode, if enabled.	All

3.2.1.3 3DSTATE_VS [DevSNB+]

3DSTATE_VS [DevSNB+]			
Project:	[DevSNB+]	**Length Bias:**	2

For **[DevSNB+]**, the state used by VS is defined with this inline state packet.

DWord	Bit	Description
0	31:29	Command Type Default Value: 3h GFXPIPE Format: OpCode
	28:27	Command SubType Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	3D Command Opcode Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	3D Command Sub Opcode Default Value: 10h 3DSTATE_VS Format: OpCode
	15:8	Reserved Project: All Format: MBZ
	7:0	DWord Length Default Value: 4h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:6	Kernel Start Pointer Project: All Address: InstructionBaseOffset[31:6] Surface Type: Kernel This field specifies the starting location (1st core instruction) of the kernel program run by threads spawned by this FF unit. It is specified as a 64-byte-granular offset from the Instruction Base Address. This field is ignored if VS Function Enable is DISABLED.
	5:0	Reserved Project: All Format: MBZ

3DSTATE_VS [DevSNB+]

2	31	Single Vertex Dispatch Project: All Format: U1 enumerated type FormatDesc This field can be used to force single vertex SIMD4x2 VS threads. Value / Name / Description / Project 0h / Multiple / Dual vertex SIMD4x2 thread dispatches are allowed. / All 1h / Single / Single vertex SIMD4x2 thread dispatches are forced. / All
	30	Vector Mask Enable (VME) Project: All Format: U1 Enumerated type When SPF=0, VME specifies which mask to use to initialize the initial channel enables. When SPF=1, VME specifies which mask to use to generate execution channel enables. Value / Name / Description / Project 0h / Dmask / Channels are enabled based on the dispatch mask / All 1h / Vmask / Channels are enabled based on the vector mask / All
	29:27	Sampler Count Project: All Format: U3 FormatDesc Specifies how many samplers (in multiples of 4) the vertex shader 0 kernel uses. Used only for prefetching the associated sampler state entries. This field is ignored if VS Function Enable is DISABLED. Value / Name / Description / Project 0h / No Samplers / no samplers used / All 1h / 1-4 Samplers / between 1 and 4 samplers used / All 2h / 5-8 Samplers / between 5 and 8 samplers used / All 3h / 9-12 Samplers / between 9 and 12 samplers used / All 4h / 13-16 Samplers / between 13 and 16 samplers used / All
	26	Reserved Project: All Format: MBZ

3DSTATE_VS [DevSNB+]

25:18 — Binding Table Entry Count

Project: All

Format: U8 FormatDesc

Range [0,255]

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

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

This field is ignored if VS Function Enable is DISABLED.

17 — Thread Priority

Project: All

Format: U1 Enumerated type

Specifies the priority of the thread for dispatch:

This field is ignored if VS Function Enable is DISABLED.

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

16 — Floating Point Mode

Project: All

Format: U1 enumerated type FormatDesc

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

This field is ignored if VS Function Enable is DISABLED.

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

15:14 — Reserved Project: All Format: MBZ

13 — Illegal Opcode Exception Enable

Project: All

Format: Enable FormatDesc

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

This field is ignored if VS Function Enable is DISABLED.

12:11 — Reserved Project: All Format: MBZ

3DSTATE_VS [DevSNB+]		
	10:8	Reserved Project: All Format: MBZ
	7	Software Exception Enable Project: All Format: Enable FormatDesc This bit gets loaded into EU CR0.1[13] (note the bit # difference). See Exceptions and ISA Execution Environment. This field is ignored if VS Function Enable is DISABLED.
	6:0	Reserved Project: All Format: MBZ
3	31:10	Scratch Space Base Offset Project: All Address: GeneralStateOffset[31:10] Surface Type: ScratchSpace Specifies the starting location of the scratch space area allocated to this FF unit as a 1K-byte aligned offset from the General State Base Address. If required, each thread spawned by this FF unit will be allocated some portion of this space, as specified by Per-Thread Scratch Space. The computed offset of the thread-specific portion will be passed in the thread payload as Scratch Space Offset. The thread is expected to utilize “stateless” DataPort read/write requests to access scratch space, where the DataPort will cause the General State Base Address to be added to the offset passed in the request header. This field is ignored if VS Function Enable is DISABLED.
	9:4	Reserved Project: All Format: MBZ
	3:0	Per-Thread Scratch Space Project: All Format: U4 power of 2 Bytes over 1K Bytes FormatDesc Range [0,11] indicating [1K Bytes, 2M Bytes] Specifies the amount of scratch space to be allocated to each thread spawned by this FF unit. The driver must allocate enough contiguous scratch space, starting at the Scratch Space Base Pointer, to ensure that the Maximum Number of Threads can each get Per-Thread Scratch Space size without exceeding the driver-allocated scratch space. This field is ignored if VS Function Enable is DISABLED. Programming Notes — Project This amount is available to the kernel for information only. It will be passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port will ignore it. — All
4	31:25	Reserved Project: All Format: MBZ

3DSTATE_VS [DevSNB+]		
	24:20	Dispatch GRF Start Register for URB Data Project: DevSNB+ Format: U5 FormatDesc Range [0,31] indicating GRF [R0,R31] Specifies the starting GRF register number for the URB portion (Constant + Vertices) of the thread payload. This field is ignored if VS Function Enable is DISABLED.
	19:17	Reserved Project: All Format: MBZ
	16:11	Vertex URB Entry Read Length Project: All Format: U6 FormatDesc Range [1,63] Specifies the number of pairs of 128-bit vertex elements to be passed into the payload for each vertex. This field is ignored if VS Function Enable is DISABLED. For SIMD4x2 dispatch, each vertex element requires one GRF of payload data, therefore the number of GRFs with vertex data will be double the value programmed in this field. **Programming Notes** Project It is UNDEFINED to set this field to 0 indicating no Vertex URB data to be read and passed to the thread.
	10	Reserved Project: All Format: MBZ
	9:4	Vertex URB Entry Read Offset Project: All Format: U6 FormatDesc Range [0,63] Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB before being included in the thread payload. This offset applies to all Vertex URB entries passed to the thread. This field is ignored if VS Function Enable is DISABLED.
	3:0	Reserved Project: All Format: MBZ

3DSTATE_VS [DevSNB+]		
5	31:25	Maximum Number of Threads Project: DevSNB Format: U7 representing thread count - 1 FormatDesc Range DevSNB: [0,59] indicating thread count of [1,60] Specifies the maximum number of simultaneous threads allowed to be active. Used to avoid using up the scratch space, or to avoid potential deadlock. This field is ignored if VS Function Enable is DISABLED.
	24:11	Reserved Project: Format: MBZ
	10	Statistics Enable Project: All Format: Enable FormatDesc If ENABLED, this FF unit will engage in statistics gathering. See the Statistics Gathering section later in this chapter. If DISABLED, statistics information associated with this FF stage will be left unchanged. [DevSNB] This field is effectively ignored if VS Function Enable is DISABLED.
	9:3	Reserved Project: All Format: MBZ
	2	Reserved Project: Format: MBZ
	1	Vertex Cache Disable Project: All Format: Disable FormatDesc This bit controls the operation of the Vertex Cache. This field is always used. If the Vertex Cache is DISABLED and the VS Function is ENABLED, the Vertex Cache is not used and all incoming vertices will be passed to VS threads. If the Vertex Cache is ENABLED and the VS Function is ENABLED, incoming vertices that do not hit in the Vertex Cache will be passed to VS threads. If the Vertex Cache is ENABLED and the VS Function is DISABLED, input vertices that miss in the Vertex Cache will be assembled and written to the URB, though pass thru the VS stage unmodified (not shaded). The Vertex Cache is invalidated whenever the Vertex Cache becomes DISABLED , whenever the VS Function Enable toggles, between 3DPRIMITIVE commands and between instances within a 3DPRIMITIVE command. **Programming Notes** **Project** See the Vertex Caching section (above) for details on implicit vertex cache disabling. All

3DSTATE_VS [DevSNB+]		
	0	VS Function Enable Project: All Format: Enable FormatDesc If ENABLED, VS threads may be spawned to process VF-generated vertices before the resulting vertices are passed down the pipeline. If DISABLED, VF-generated vertices will pass thru the VS function and sent down the pipeline unmodified. The Vertex Cache is still available in this mode, if enabled. This field is always used. [DevSNB: A,B] VS function Enable must always be disabled.
6	31:17	Reserved Project: Format: MBZ
	20:16	Reserved
	15:8	Reserved
	7:0	Reserved

3.2.1.4 3DSTATE_CONSTANT_VS [DevSNB]

3DSTATE_CONSTANT_VS			
Project:	DevSNB	**Length Bias:**	2

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

[DevSNB A] All memory accesses are to GGTT address space, independent of the PPGTT mode bit in GFX_MODE

DWord	Bit	Description
0	31:29	Command Type Default Value: 3h GFXPIPE Format: OpCode
	28:27	Command SubType Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	3D Command Opcode Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	3D Command Sub Opcode Default Value: 15h 3DSTATE_CONSTANT_VS Format: OpCode
	15	Buffer 3 Valid Project: All Format: Enable This field enables buffer 3

3DSTATE_CONSTANT_VS		
	14	Buffer 2 Valid Project: All Format: Enable This field enables buffer 2
	13	Buffer 1 Valid Project: All Format: Enable This field enables buffer 1
	12	Buffer 0 Valid Project: All Format: Enable This field enables buffer 0
	11:8	Constant Buffer Object Control State Project: All Format: MEMORY_OBJECT_CONTROL_STATE FormatDesc Specifies the memory object control state for all constant buffers defined in this command. [DevSNB:A] Only bits 11 and 10 can be used (GFDT)
	7:0	DWord Length Default Value: 3h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:5	Pointer to VS Constant Buffer 0 Project: All Address: DynamicStateOffset[31:5] or GraphicsAddress[31:5] Surface Type: ConstantBuffer This field points to the location of VS Constant Buffer 0. The state of **INSTPM<CONSTANT_BUFFER Address Offset Disable>** determines whether the Dynamic State Base Address is added to this pointer. Programming Notes Constant buffers must be allocated in linear (not tiled) graphics memory.
	4:0	VS Constant Buffer 0 Read Length Project: All Format: U5 read length – 1 This field specifies the length of the constant data to be loaded from memory in 256-bit units minus one. Programming Notes The sum of all four read length fields (each incremented to represent the actual read length) must be less than or equal to 32

3DSTATE_CONSTANT_VS		
2	31:5	Pointer to VS Constant Buffer 1 Project: All Address: GraphicsAddress[31:5] Surface Type: ConstantBuffer This field points to the location of VS Constant Buffer 1. Programming Notes Constant buffers must be allocated in linear (not tiled) graphics memory.
	4:0	VS Constant Buffer 1 Read Length Project: All Format: U5 read length – 1 This field specifies the length of the constant data to be loaded from memory in 256-bit units minus one. Programming Notes The sum of all four read length fields (each incremented to represent the actual read length) must be less than or equal to 32
3	31:5	Pointer to VS Constant Buffer 2 Project: All Address: GraphicsAddress[31:5] Surface Type: ConstantBuffer This field points to the location of VS Constant Buffer 2. Programming Notes Constant buffers must be allocated in linear (not tiled) graphics memory.
	4:0	VS Constant Buffer 2 Read Length Project: All Format: U5 read length – 1 This field specifies the length of the constant data to be loaded from memory in 256-bit units minus one. Programming Notes The sum of all four read length fields (each incremented to represent the actual read length) must be less than or equal to 32

3DSTATE_CONSTANT_VS		
4	31:5	Pointer to VS Constant Buffer 3 Project: All Address: GraphicsAddress[31:5] Surface Type: ConstantBuffer This field points to the location of VS Constant Buffer 3. Programming Notes Constant buffers must be allocated in linear (not tiled) graphics memory.
	4:0	VS Constant Buffer 3 Read Length Project: All Format: U5 read length – 1 This field specifies the length of the constant data to be loaded from memory in 256-bit units minus one. Programming Notes The sum of all four read length fields (each incremented to represent the actual read length) must be less than or equal to 32

3.2.2 Input Vertices

Refer to *3D Overview* for a description of the vertex information input to the VS stage.

3.3 SIMD4x2 VS Thread Request Generation

This section describes SIMD4x2 thread request generation.

The following discussion assumes the VS Function is ENABLED.

When the Vertex Cache is disabled, the VS unit will pass each pair of incoming vertices to a VS thread. Under certain circumstances (e.g., prior to a state change or pipeline flush) the VS unit will spawn a VS thread to process a single vertex. Note that, in this case, the “unused” vertex slot will be “disabled” via the Execution Mask provided by the VS unit to the subsystem as part of the thread dispatch (See ISA doc). The VS thread will in itself be unaware of the single-vertex case, and therefore a single VS kernel can be used to process one or two vertices. (The performance of single-vertex processing will roughly equal the two-vertex case).

When the Vertex Cache is enabled, the VF unit will detect vertices that hit in the cache and mark these vertices so that they will bypass VS thread processing and be output via a reference to the cached VUE. The VS unit will keep track of these cache-hit vertices as it proceeds to process cache-miss vertices. The VS unit guarantees that vertices will exit the unit in the order they are received. This may require the VS unit to issue single-vertex VS threads to process a cache-miss vertex that has yet to be paired up with another cache-miss vertex (if this condition is preventing the VS unit from producing any output).

3.3.1 Thread Payload

The following table describes the payload delivered to VS threads.

Table 12. VS Thread Payload (SIMD4x2)

DWord	Bit	Description
R0.7	31	Snapshot Flag
	30:0	Reserved
R0.6	31:24	Reserved
	23:0	**Thread ID:** This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	**Scratch Space Offset:** Specifies the of the scratch space allocated to the thread, specified as a 1KB-granular offset from the **General State Base Address**. See **Scratch Space Base Offset** description in VS_STATE. (See *3D Pipeline* for further description on scratch space allocation). Format = GeneralStateOffset[31:10]
	9	Reserved
	8:0	**FFTID:** This ID is assigned by the FF unit and used to identify the thread within the set of outstanding threads spawned by the FF unit. Format: Reserved for HW Implementation Use.
R0.4	31:5	**Binding Table Pointer.** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved
R0.3	31:5	**Sampler State Pointer.** Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the **General State Base Address** or **Dynamic State Base Address**. Format = DynamicStateOffset[31:5] **[DevSNB+]**
	4	Reserved

DWord	Bit	Description
	3:0	**Per Thread Scratch Space:** Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two will be raised to (over determine the amount of scratch space). (See *3D Pipeline* for further description). Format = U4 power of two (in excess of 10) Range = [0,11] indicating [1K Bytes, 2M Bytes]
R0.2	31:0	Reserved : delivered as zeros (reserved for message header fields)
R0.1	31:16	Reserved
	15:0	**URB Return Handle 1:** This is the 64B-aligned URB offset where the EU's upper channels (DWords 7:4) results are to be stored. If only one vertex is to be processed (shaded) by the thread, this field will effectively be ignored (no results are stored for these channels, as controlled by the thread's Channel Mask). (See *Generic FF Unit* for further description). Format: U9 opaque handle **[Pre-DevILK]** Format: U10 opaque handle **[DevILK]** Format: U11 64B-aligned offset [DevSmallGT] Format: U12 64B-aligned offset [DevSNB+]
R0.0	31:16	Reserved
	15:0	**URB Return Handle 0:** This is the 64B-aligned URB offset where the EU's lower channels (DWords 3:0) results are to be stored. (See *Generic FF Unit* for further description). Format: U12 64B-aligned offset [DevSNB+]
[Varies] optional	255:0	Constant Data (optional) : **[DevSNB+]:** Some amount of constant data (possible none) can be extracted from the push constant buffer (PCB) and passed to the thread following the R0 Header. The amount of data provided is defined by the sum of the read lengths in the last 3DSTATE_CONSTANT_VS command (taking the buffer enables into account). The Constant Data arrives in a non-interleaved format.

DWord	Bit	Description
Varies	255:0	**Vertex Data :** Data from (possibly) one or (more typically) two Vertex URB Entries is passed to the thread in the thread payload. The **Vertex URB Entry Read Offset** and **Vertex URB Entry Read Length** state variables define the regions of the URB entries that are read from the URB and passed in the thread payload. These SVs can be used to provide a subset of the URB data as required by SW. The vertex data is laid out in the thread header in an interleaved format. The lower DWords (0-3) of these GRF registers always contain data from a Vertex URB Entry. The upper DWords (4-7) may contain data from another Vertex URB Entry. This allows two vertices to be processed (shaded) in parallel SIMD8 fashion. The VS kernel is not aware of the validity of the upper vertex.

3.4 SIMD4x2 VS Thread Execution

This section describes SIMD4x2 thread execution.

A VS kernel (with one exception mentioned below) assumes it is to operate on two vertices in parallel. Input data is either passed directly in the thread payload (including the input vertex data) or indirectly via pointers passed in the payload.

Refer to *ISA* chapters for specifics on writing kernels that operate in SIMD4x2 fashion.

Refer to 3D Pipeline Stage Overview (*3D Overview*) for information on FF-unit/Thread interactions.

In the (unlikely) event that the VS kernel needs to determine whether it is processing one or two vertices, the kernel can compare the **URB Return Handle 0** and **URB Return Handle 1** fields of the thread payload. These fields will be different if two vertices are being processed, and identical if one vertex is being processed. An example of when this test may be required is if the kernel outputs some vertex-dependent results into a memory buffer – without the test the single vertex case might incorrectly output two sets of results. Note that this is not the case for writing the URB destinations, as the Execution Mask will prevent the write of an undefined output.

Prior to sending an End Of Thread, the kernel must dispatch a write commit cycle, if there were any previous writes to memory that had caused no dependency checks.

3.4.1 Vertex Output

VS threads must always write the destination URB handles passed in the payload. VS threads are not permitted to request additional destination handles. Refer to 3D Pipeline Stage Overview (*3D Overview*) for details on how destination vertices are written and any required contents/formats.

3.4.2 Thread Termination

VS threads must signal thread termination, in all likelihood on the last message output to the URB shared function. Refer to the *ISA* doc for details on End-Of-Thread indication.

3.5 Primitive Output

The VS unit will produce an output vertex reference for every input vertex reference received from the VF unit, in the order received. The VS unit simply copies the PrimitiveType, StartPrim, and EndPrim information associated with input vertices to the output vertices, and does not use this information in any way. Neither does the VS unit perform any readback of URB data.

3.6 Other VS Functions

3.6.1 Statistics Gathering

The VS stage tracks a single pipeline statistic, the number of times a vertex shader is executed. A vertex shader is executed for each vertex that is fetched on behalf of a 3DPRIMITIVE command, unless the shaded results for that vertex are already available in the vertex cache. If the **Statistics Enable** bit in VS_STATE is set, the VS_INVOCATION_COUNT Register (see Memory Interface Registers in Volume Ia, *GPU*) will be incremented for *each vertex* that is dispatched to a VS thread. This counter will often need to be incremented by 2 for each thread invoked since 2 vertices are dispatched to one VS thread in the general case.

[DevSNB:B0]: When **VS Function Enable** is DISABLED and **Statistics Enable** is ENABLED, VS_INVOCATION_COUNT will increment by one for every vertex that passes through the VS stage, even though no VS threads are spawned.

4. Geometry Shader (GS) Stage

4.1 GS Stage Overview

The GS stage of the 3D Pipeline is used to convert objects within incoming primitives into new primitives through use of a spawned thread. When enabled, the GS unit buffers incoming vertices, assembles the vertices of each individual object within the primitives, and passes these object vertices (along with other data) to the subsystem for processing by a GS thread.

When the GS stage is disabled, vertices flow through the unit unmodified.

Refer to the *Common 3D FF Unit Functions* subsection in the *3D Pipeline* chapter for a general description of a 3D Pipeline stage, as much of the GS stage operation and control falls under these "common" functions. That is, most stage state variables and GS thread payload parameters are described in *3D Pipeline*, and although they are listed here for completeness, that chapter provides the detailed description of the associated functions.

Refer to this chapter for an overall description of the GS stage, and any exceptions the GS stage exhibits with respect to common FF unit functions.

4.2 GS Stage Input

As a stage of the 3D pipeline, the GS stage receives inputs from the previous VS stage. Refer to *3D Pipeline* for an overview of the various types of input to a 3D Pipeline stage. The remainder of this subsection describes the inputs specific to the GS stage.

4.2.1 State

4.2.1.1 3DSTATE_GS_SVB_INDEX [DevSNB]

The 3DSTATE_GS_SVB_INDEX instruction is used to program geometry shader streamed vertex buffer indexes or the Internal Vertex Count state register.

Four independent index values are supported. Each instance of this instruction programs one of the indexes, selected by the **Index Number** field. All four indexes are delivered to the geometry shader thread, and kernel code is responsible for using the correct index for each data port message.

This instruction is treated like non-pipelined state, thus a pipeline flush is executed before the indexes are changed.

DWord	Bit	Description
0	31:29	**Instruction Type =** 3D_INSTRUCTION = 3h
	28:16	3D Instruction Opcode = 3DSTATE_GS_SVB_INDEX GFXPIPE[28:27 = 3h, 26:24 = 1h; 23:16 = 0Bh]
	15:0	**DWord Length** (Excludes DWords 0,1) = **[DevSNB+]**: 2
1	31	Reserved: MBZ
	30:29	**Index Number:** Selects which index is to be loaded. Format = U2 Range = [0,3]
	28:0	Reserved : MBZ
2	31:0	Streamed Vertex Buffer Index This field contains the value of the specified index, or the pointer to the index depending on the **Index Loading Mode** field. Format = U32 index Range = $[0,2^{27}-1]$
3 DevSNB + only	31:0	**[DevSNB+]: Maximum Index.** This field specifies the maximum value of the selected SVBI, enforced by the device. Software should set this field to one past the last valid vertex index, so that the clamped value can be used as a vertex count (see **Internal Vertex Count** in 3DPRIMITIVE. Format = U32 index Range = $[0,2^{27}]$

3DSTATE_GS_SVB_INDEX

Project:	[DevSNB]	**Length Bias:**	2

The 3DSTATE_GS_SVB_INDEX instruction is used to program geometry shader streamed vertex buffer indexes.

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

3DSTATE_GS_SVB_INDEX

	7:0	DWord Length Default Value: 1h Excludes DWord (0,1) Format: =n Total Length - 2 Project: [preDevIL]
	7:0	DWord Length Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2 Project: [DevIL+]
1	31	Reserved Project: All Format: MBZ
	30:29	Index Number Project: All Format: U2 Index of SVBI register Range 0-3 Selects which index is to be loaded. For DevSNB+, this field is alternatively used to select the SVBI register whose value is to be loaded into the Internal Vertex Count state register (see Load Internal Vertex Count field below).
	28:1	Reserved Project: All Format: MBZ
	0	Load Internal Vertex Count Project: DevSNB:B Format: Boolean FormatDesc If set, the current contents of the selected SVBI register is loaded into the Internal Vertex Count state register. If clear, the selected SVBI register/max-index is loaded from the **Streamed Vertex Buffer Index** and **Maximum Index** fields. The Internal Vertex Count register is left unmodified.

Value	Name	Description	Project
0h	Disable		DevSNB:B
1h	Enable		DevSNB:B

3DSTATE_GS_SVB_INDEX

2	31:0	**Streamed Vertex Buffer Index (SVBI)** Project: All Format: U32 FormatDesc Range 0..2^27-1 This field contains the value to be loaded into the SVBI register selected by Index Number. **Programming Notes** [DevSNB] If a buffer is not enabled then the SVBI must be set to 0x0 in order to not cause overflow in that SVBI. [DevSNB] The SVBI value must never be programmed to a value greater than the Maximum Index
3	31:0	**Maximum Index** Project: DevSNB Format: U32 Index into an SVB Range 0..2^27 This field specifies the maximum value of the selected SVBI, enforced by the device. Software should set this field to one past the last valid vertex index, so that the clamped value can be used as a vertex count (see Internal Vertex Count in 3DPRIMITIVE. **Programming Notes** Once and SVBI reaches the Maximum Index then all SVBI values will discontinue to increment. If a buffer is not enabled then the MaxSVBI must be set to 0xFFFFFFFF in order to not cause overflow in that SVBI.

4.2.1.2 3DSTATE_GS [DevSNB]

For **[DevSNB]**, the state used by GS is defined with this inline state packet.

3DSTATE_GS			
Project:	[DevSNB]	**Length Bias:**	2
Controls the GS stage hardware.			

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

3DSTATE_GS

2	31	**Single Program Flow (SPF)** Project: All Specifies the initial condition of the kernel program as either a single program flow (SIMDnxm with m = 1) or as multiple program flows (SIMDnxm with m > 1). See CR0 description in ISA Execution Environment. Value / Name / Description / Project 0h / Reserved / / All 1h / Enable / Single Program Flow enabled / All
	30	Vector Mask Enable (VME) Project: All Format: U1 enumerated type FormatDesc When SPF=0, VME specifies which mask to use to initialize the initial channel enables. When SPF=1, VME specifies which mask to use to generate execution channel enables.

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

3DSTATE_GS

	29:27	Sampler Count Project: All Format: U3 Specifies how many samplers (in multiples of 4) the geometry shader kernel uses. Used only for prefetching the associated sampler state entries. Value / Name / Description / Project 0h / No Samplers / no samplers used / All 1h / 1-4 Samplers / between 1 and 4 samplers used / All 2h / 5-8 Samplers / between 5 and 8 samplers used / All 3h / 9-12 Samplers / between 9 and 12 samplers used / All 4h / 13-16 Samplers / between 13 and 16 samplers used / All 5h-7h / Reserved / Reserved / All
	26	Reserved Project: All Format: MBZ
	25:18	Binding Table Entry Count Project: All Format: U8 Specifies how many binding table entries the kernel uses. Used only for prefetching of the binding table entries and associated surface state. **Note**: For kernels using a large number of binding table entries, it may be wise to set this field to zero to avoid prefetching too many entries and thrashing the state cache.
	17	Thread Priority Project: All Specifies the priority of the thread for dispatch Value / Name / Description / Project 0h / Normal Priority / Normal Priority / All 1h / High Priority / High Priority / All

3DSTATE_GS		
	16	Floating Point Mode Project: All Specifies the initial floating point mode used by the dispatched thread. Value / Name / Description / Project 0h / IEEE-754 / Use IEEE-754 Rules / All 1h / alternate / Use alternate rules / All
	15:14	Reserved Project: All Format: MBZ
	13	Illegal Opcode Exception Enable Project: All Format: Enable This bit gets loaded into EU CR0.1[12] (note the bit # difference). See *Exceptions* and *ISA Execution Environment*.
	12	Reserved Project: All Format: MBZ
	11	Mask Stack Exception Enable Project: All Format: Enable This bit gets loaded into EU CR0.1[11]. See *Exceptions* and *ISA Execution Environment*.
	10:8	Reserved Project: All Format: MBZ
	7	Software Exception Enable Project: All Format: Enable This bit gets loaded into EU CR0.1[13] (note the bit # difference). See *Exceptions* and *ISA Execution Environment*.
	6:0	Reserved Project: All Format: MBZ
3	31:10	Scratch Space Base Pointer Project: All Format: GeneralStateOffset[31:10] Specifies the location of the scratch space area allocated to this FF unit, specified as a 1KB-granular offset from the General State Base Address. If required, each thread spawned by this FF unit will be allocated some portion of this space, as specified by Per-Thread Scratch Space.
	9:4	Reserved Project: All Format: MBZ

3DSTATE_GS		
	3:0	Per-Thread Scratch Space Project: All Format: U4 power of 2 Bytes over 1K Bytes FormatDesc Range [0,11] indicating [1K Bytes, 2M Bytes] Specifies the amount of scratch space to be allocated to each thread spawned by this FF unit. The driver must allocate enough contiguous scratch space, starting at the Scratch Space Base Pointer, to ensure that the Maximum Number of Threads can each get Per-Thread Scratch Space size without exceeding the driver-allocated scratch space.
4	31:17	Reserved Project: All Format: MBZ
	16:11	Vertex URB Entry Read Length Project: All Format: U6 FormatDesc Range [1,63] Specifies the amount of URB data read and passed in the thread payload <u>for each Vertex URB entry</u>, in 256-bit register increments. It is UNDEFINED to set this field to 0 indicating no Vertex URB data to be read and passed to the thread.
	10	Reserved Project: All Format: MBZ
	9:4	Vertex URB Entry Read Offset Project: All Format: U6 FormatDesc Range [0,63] Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB before being included in the thread payload. This offset applies to all Vertex URB entries passed to the thread.

	3:0	Dispatch GRF Start Register for URB Data Project: All Format: U4 FormatDesc Range [0,15] indicating GRF [R0,R15] Specifies the starting GRF register number for the URB portion (Constant + Vertices) of the thread payload.
5	31:25	Maximum Number of Threads Project: All Format: U7 thread count – 1 Range DevSNB: [0,59] indicating thread count of [1,60] Specifies the maximum number of simultaneous threads allowed to be active. Used to avoid using up the scratch space, or to avoid potential deadlock. Programming Notes: A URB_FENCE command must be issued subsequent to any change to the value in this field and before any subsequent pipeline processing (e.g., via 3DPRIMITIVE or CONSTANT_BUFFER). See Graphics Processing Engine (Command Ordering Rules) [DevSNB] Maximum Number of Threads valid range is [0,27] when Rendering Enabled bit is set.
	24:11	Reserved Project: All Format: MBZ
	10	GS Statistics Enable Project: All Format: Enable This bit controls whether GS-unit-specific statistics register(s) can be incremented. Value / Name / Description / Project 0h / Disable / GS_INVOCATIONS_COUNT and GS_PRIMITIVES_COUNT cannot increment / All 1h / Enable / GS_INVOCATIONS_COUNT and GS_PRIMITIVES_COUNT can increment / All

	9	SO Statistics Enable Project: All Format: Enable This bit controls whether certain StreamOutput statistics register(s) can be incremented. Value / Name / Description / Project 0h / Disable / SO_NUM_PRIMS_WRITTEN and SO_PRIM_STORAGE_NEEDED cannot increment / All 1h / Enable / SO_NUM_PRIMS_WRITTEN and SO_PRIM_STORAGE_NEEDED can increment / All
	8	Rendering Enabled Project: All Format: U1 This state bit is used to indicate whether or not the GS thread will be allocating and outputting VUE handles for rendering. This bit must be set if the thread will attempt to allocate a handle. If clear, the GS thread must not allocate handles (e.g., when only performing stream output without concurrent rendering).
	7:0	Reserved Project: All Format: MBZ
6	31	Reserved Project: All Format: MBZ

3DSTATE_GS		
	30	Reorder Enable Project: All Format: Enable This bit controls whether the GS unit reorders TRISTRIP/TRISTRIP_REV vertices passed in the GS thread payload. If ENABLED, the GS unit will reorder the vertices for “odd-numbered” triangles originating from TRISTRIP topologies and “even-numbered” triangles originating from TRISTRIP_REV topologies. (Note that the first triangle is considered “triangle 0”, which is even-numbered). With respect to the PrimType passed in the GS thread payload, the GS unit passes TRISTRIP when the vertices <u>are not</u> reordered, and TRISTRIP_REV when the vertices <u>are</u> reordered (regardless of whether a TRISTRIP or TRISTRIP_REV topology was being processed) If DISABLED, TRISTRIP/TRISTRIP_REV vertices are not reordered, and always passed in the order they are received from the pipeline. The GS unit will still toggle PrimType on alternating (as described above) so that the GS thread can perform the reordering internally (or do whatever is necessary to account for the non-reordering of its input).
	29	Discard Adjaceny Project: All Format: Enable When set, adjacent vertices <u>will not be passed</u> in the GS payload when objects with adjacency are processed. Instead, only the non-adjacent vertices will be passed in the same fashion as the without-adjacency form of the primitive. Software should set this bit whenever a GS kernel is used that <u>does not expect</u> adjacent vertices. This allows both with-adjacency/without-adjacency variants of the primitive to be submitted to the pipeline (via 3DPRIMITIVE) – the GS unit will silently discard any adjacent vertices and present the GS thread with only the internal object. When clear, adjacent vertices <u>will be passed</u> to the GS thread, as dictated by the incoming primitive type. Software should only clear this bit when a GS kernel is used that <u>does expect</u> adjacent vertices. E.g., if the GS kernel is compiled to expect a TRIANGLE_ADJ object, software must clear this bit. Software should also clear this bit if the GS kernel expects a POINT object (which doesn’t have a with-adjacency variant). This bit is used to provide limited compatibility between submitted primitive types and the object type expected by the GS kernel. The only hardware assistance is to allow the submission of a with-adjacency variant of a primitive when operating with a GS kernel that expects the without-adjacency variant of the object. (E.g., when the GS kernel is compiled to expect a TRIANGLE object, software should set this bit just in case a TRILIST_ADJ is submitted to the pipeline.) Note that the GS unit is otherwise not aware of the object type that is expected by the GS kernel. It is up to software to ensure that the submitted primitive type (in 3DPRIMITIVE) is otherwise compatible with the object type expected by the GS kernel. (E.g., if the GS kernel expects a LINE_ADJ object, only LINELIST_ADJ or LINESTRIP_ADJ should be submitted, otherwise the GS kernel will produce

		unpredictable results.) Also note that it is possible to craft a GS kernel which can accept any object type that's thrown at it by first examining the PrimType passed in the payload and then using this info to correctly interpret the number of vertices passed in the payload.
	28	SVBI Payload Enable Project: All Format: Enable This field controls whether the optional R1 header phase containing the Streamed Vertex Buffer Indices is delivered. [DevSNB+]: The optional R1 header phase now also contains the Maximum Index Values for the SVBs (in previously-reserved DWords).
	27	SVBI Post-Increment Enable Project: All Format: Enable This bit should be set whenever the GS thread is performing <u>only</u> the SO function (no GS, no Render). Setting this bit allows the GS FF unit to post-increment the SVBI values after GS threads are dispatched. This allows the threads to complete without the need for further synchronization. The increment value is specified by SVBI Post-Increment Value. If this bit is clear, the GS thread must use the FF_SYNC message to report the amount of data it will output and receive the appropriate (synchronized) SVBI values in the writeback. This is required whenever the GS API function is required and software cannot guarantee that the GS shader will output a constant amount of output (vertices). **Programming Note:** Since the GS threads are provided with overflow-clamped SVBI inputs, they are always responsible for overflow detection given those inputs.
	26	Reserved Project: All Format: MBZ
	25:16	SVBI Post-Increment Value Project: All Format: U10 If SVBI Post-Increment Enable is set, all the SVBI state registers will be incremented by this value after the dispatch of every GS thread. If SVBI Post-Increment Enable is clear, this field is ignored.

3DSTATE_GS		
	15	GS Enable Project: All Format: Enable FormatDesc Specifies whether the GS function is enabled or disabled (pass-through). **Programming Note:** When enabling the GS stage that may generate incomplete objects, the CLIP stage also needs to be ENABLED in order to filter out any incomplete objects. See *Clipper* chapter.
	14:0	Reserved Project: All Format: MBZ

4.2.1.3 3DSTATE_CONSTANT_GS [DevSNB]

3DSTATE_CONSTANT_GS			
Project:	[DevSNB]	**Length Bias:**	2

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

[DevSNB A] All memory accesses are to GGTT address space, independent of the PPGTT mode bit in GFX_MODE

Programming Note:

It is invalid to program this command mroe then once between 3D_PRIMITIVE commands.

DWord	Bit	Description
0	31:29	Command Type Default Value: 3h GFXPIPE Format: OpCode
	28:27	Command SubType Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	3D Command Opcode Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	3D Command Sub Opcode Default Value: 16h 3DSTATE_CONSTANT_GS Format: OpCode
	15	Buffer 3 Valid Project: All Format: Enable This field enables buffer 3
	14	Buffer 2 Valid Project: All Format: Enable This field enables buffer 2
	13	Buffer 1 Valid Project: All Format: Enable This field enables buffer 1

3DSTATE_CONSTANT_GS		
	12	**Buffer 0 Valid** Project: All Format: Enable This field enables buffer 0
	11:8	**Constant Buffer Object Control State** Project: All Format: MEMORY_OBJECT_CONTROL_STATE FormatDesc Specifies the memory object control state for all constant buffers defined in this command.
	7:0	**DWord Length** Default Value: 3h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:5	**Pointer to GS Constant Buffer 0** Project: All Address: DynamicStateOffset[31:5] or GraphicsAddress[31:5] Surface Type: ConstantBuffer This field points to the location of GS Constant Buffer 0. The state of **INSTPM<CONSTANT_BUFFER Address Offset Disable>** determines whether the Dynamic State Base Address is added to this pointer. **Programming Notes** Constant buffers must be allocated in linear (not tiled) graphics memory.

3DSTATE_CONSTANT_GS

	4:0	**GS Constant Buffer 0 Read Length** Project: All Format: U5 (read length – 1) FormatDesc This field specifies the length of the constant data to be loaded from memory in 256-bit units minus one. **Programming Notes** The sum of all four read length fields (each incremented to represent the actual read length) must be less than or equal to 64
2	31:5	**Pointer to GS Constant Buffer 1** Project: All Address: GraphicsAddress[31:5] Surface Type: ConstantBuffer This field points to the location of GS Constant Buffer 1. **Programming Notes** Constant buffers must be allocated in linear (not tiled) graphics memory.
	4:0	**GS Constant Buffer 1 Read Length** Project: All Format: U5 (read length – 1) FormatDesc This field specifies the length of the constant data to be loaded from memory in 256-bit units minus one. **Programming Notes** The sum of all four read length fields (each incremented to represent the actual read length) must be less than or equal to 64

3DSTATE_CONSTANT_GS		
3	31:5	Pointer to GS Constant Buffer 2 Project: All Address: GraphicsAddress[31:5] Surface Type: ConstantBuffer This field points to the location of GS Constant Buffer 2. Programming Notes Constant buffers must be allocated in linear (not tiled) graphics memory.
	4:0	GS Constant Buffer 2 Read Length Project: All Format: U5 (read length – 1) FormatDesc This field specifies the length of the constant data to be loaded from memory in 256-bit units minus one. Programming Notes The sum of all four read length fields (each incremented to represent the actual read length) must be less than or equal to 64
4	31:5	**Pointer to GS Constant Buffer 3** Project: All Address: GraphicsAddress[31:5] Surface Type: ConstantBuffer This field points to the location of GS Constant Buffer 3. **Programming Notes** Constant buffers must be allocated in linear (not tiled) graphics memory.

3DSTATE_CONSTANT_GS		
	4:0	GS Constant Buffer 3 Read Length Project: All Format: U5 (read length – 1) FormatDesc This field specifies the length of the constant data to be loaded from memory in 256-bit units minus one. **Programming Notes** The sum of all four read length fields (each incremented to represent the actual read length) must be less than or equal to 64

4.3 Object Staging

The GS unit's Object Staging Buffer (OSB) accepts primitive topologies as a stream of incoming vertices, and spawns a thread for each individual object within the topology.

4.4 GS Thread Request Generation

4.4.1 Object Vertex Ordering [DevSNB]

The following table defines the number and order of object vertices passed in the Vertex Data portion of the GS thread payload, assuming an input topology with *N* vertices. The ObjectType passed to the thread is, by default, the incoming PrimTopologyType. Exceptions to this rule (for the TRISTRIP variants) are called out.

[**DevSNB+**]: The following table also shows which vertex is selected to provide PrimitiveID (**bold, underlined vertex number**). In general, the vertex selected is the last vertex for non-adjacent prims, and the next-to-last vertex for adjacent prims. Note, however, that there are exceptions:

- reorder-enabled TRISTRIP[_REV]
- "odd-numbered" objects in TRISTRIP_ADJ

PrimTopologyType	Order of Vertices in Payload	GS Notes
<PRIMITIVE_TOPOLOGY> (N = # of vertices)	[<object#>] = (<vert#>,…); [{modified PrimType passed to thread}]	
POINTLIST	[0] = (**0**); [1] = (**1**); …; [N-2] = (**N-2**);	
POINTLIST_BF	N/A	
LINELIST (N is multiple of 2)	[0] = (0,**1**); [1] = (2,**3**); …; [(N/2)-1] = (N-2,**N-1**)	
LINELIST_ADJ (N is multiple of 4)	[0] = (0,1,**2**,3); [1] = (4,5,**6**,7); …; [(N/4)-1)] = (N-4,N-3,**N-2**,N-1)	
LINESTRIP (N >= 2)	[0] = (0,**1**); [1] = (1,**2**); …; [N-2] = (N-2,**N-1**)	
LINESTRIP_ADJ (N >= 4)	[0] = (0,1,**2**,3); [1] = (1,2,**3**,4); …; [N-4] = (N-4,N-3,**N-2**,N-1)	
LINESTRIP_BF	N/A	
LINESTRIP_CONT	Same as LINESTRIP	Handled same as LINESTRIP
LINESTRIP_CONT_BF	Same as LINESTRIP	Handled same as LINESTRIP
LINELOOP (N >= 2)	[0] = (0,**1**); [1] = (1,**2**); [N] = (N-1,**0**);	Not supported after GS.

PrimTopologyType	Order of Vertices in Payload	GS Notes
TRILIST (N is multiple of 3)	[0] = (0,1,**2**); [1] = (3,4,**5**); …; [(N/3)-1] = (N-3,N-2,**N-1**)	
RECTLIST	Same as TRILIST	Handled same as TRILIST
TRILIST_ADJ (N is multiple of 6)	[0] = (0,1,2,3,**4**,5); [1] = (6,7,8,9,**10**,11); …; [(N/6)-1] = (N-6,N-5,N-4,N-3,**N-2**,N-1)	
TRISTRIP (Reorder ENABLED) (N >= 3)	[0] = (0,1,**2**); {TRISTRIP} [1] = (1,**3**,2); {TRISTRIP_REV} [k even] = (k,k+1,**k+2**) {TRISTRIP} [k odd] = (k,**k+2**,k+1) {TRISTRIP_REV} [N-3] = (see above)	"Odd" triangles have vertices reordered though identified as TRISTRIP_REV so the thread knows this
TRISTRIP (Reorder DISABLED) (N >= 3)	[0] = (0,1,**2**) {TRISTRIP} [1] = (1,2,**3**) {TRISTRIP_REV}; … [N-3] = (N-3,N-2,**N-1**) {TRISTRIP or TRISTRIP_REV}	"Odd" triangles do not have vertices reordered, though identified as TRISTRIP_REV so the thread knows this
TRISTRIP_REV (Reorder ENABLED) (N >= 3)	[0] = (0,**2**,1) {TRISTRIP_REV}; [1] = (1,2,**3**) {TRISTRIP}; …; [k even] = (k,**k+2**,k+1) {TRISTRIP_REV} [k odd] = (k,k+1,**k+2**) {TRISTRIP} [N-3] = (see above)	"Odd" triangles have vertices reordered, though identified as TRISTRIP so the thread knows this
TRISTRIP_REV (Reorder DISABLED) (N >= 3)	[0] = (0,1,**2**) {TRISTRIP_REV} [1] = (1,2,**3**) {TRISTRIP}; …; [N-3] = (N-3,N-2,**N-1**) {TRISTRIP or TRISTRIP_REV}	"Odd" triangles do not have vertices reordered, though identified as TRISTRIP so the thread knows this

PrimTopologyType	Order of Vertices in Payload	GS Notes
TRISTRIP_ADJ (N even, N >= 6)	N = 6 or 7: [0] = (0,1,2,5,**4**,3) N = 8 or 9: [0] = (0,1,2,6,**4**,3); [1] = (2,5,**6**,7,4,0); …; N > 10: [0] = (0,1,2,6,**4**,3); [1] = (2,5,**6**,8,4,0); …; [k>1, even] = (2k,2k-2, 2k+2, 2k+6,**2k+4**, 2k+3); [k>2, odd] = (2k, 2k+3, **2k+4**, 2k+6, 2k+2, 2k-2);…; Trailing object: [(N/2)-3, even] = (N-6,N-8,N-4,N-1,**N-2**,N-3); [(N/2)-3, odd] = (N-6,N-3,**N-2**,N-1,N-4,N-8);	"Odd" objects have vertices reordered.
TRIFAN (N > 2)	[0] = (0,1,**2**); [1] = (0,2,**3**); …; [N-3] = (0, N-2, **N-1**);	Only used by OGL
TRIFAN_NOSTIPPLE	Same as TRIFAN	
POLYGON	Same as TRIFAN	
QUADLIST (N is multiple of 4)	[0] = (0,1,2,**3**); [1] = (4,5,6,**7**); …; [(N/4)-1] = (N-4,N-3,N-2,**N-1**);	Not supported after GS.
QUADSTRIP (N is multiple of 2, N >=4)	[0] = (0,1,3,2); [1] = (2,3,5,4); … ; [(N/2)-2] = (N-4,N-3,N-1,N-2);	Not supported after GS.

4.4.2 GS Thread Payload [DevSNB]

The table below shows the layout of the payload delivered to GS threads.

Refer to 3D Pipeline Stage Overview (*3D Pipeline*) for details on those fields that are common amongst the various pipeline stages.

GS Thread Payload [DevSNB]

GRF DWord	Bit	Description
R0.7	31	Snapshot Flag.
	30:0	Reserved
R0.6	31:24	Reserved
	23:0	**Thread ID.** This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	**Scratch Space Pointer.** Specifies the location of the scratch space allocated to this thread, specified as a 1KB-aligned offset from the **General State Base Address**. Format = GeneralStateOffset[31:10]
	9:8	Reserved
	7:0	**FFTID.** This ID is assigned by the fixed function unit and is relative identifier for the thread. It is used to free up resources used by the thread upon thread completion. Format: Reserved for Implementation Use
R0.4	31:5	**Binding Table Pointer:** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved
R0.3	31:5	**Sampler State Pointer.** Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the **General State Base Address** or **Dynamic State Base Address**. Format = DynamicStateOffset[31:5] **[DevSNB+]**
	4	Reserved

GRF DWord	Bit	Description
	3:0	**Per Thread Scratch Space.** Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two will be raised to (over determine the amount of scratch space). (See *Generic Pipeline Stage* for further description). Programming Notes: This amount is available to the kernel for information only. It will be passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port will ignore it. Format = U4 power of two (in excess of 10) Range = [0,11] indicating [1K Bytes, 2M Bytes]
R0.2	31:10	Reserved : delivered as zeros (reserved for message header fields)
	9	**Edge Indicator [1].** For POLYGON primitive objects, this bit indicates whether the edge from Vertex2 to Vertex0 is an exterior edge of the polygon (i.e., this is the last or only triangle of the polygon). If clear, that edge is an interior edge. The kernel can use this bit to control operations such as generating wireframe representations of polygon primitives. For all other Primitive Topology Types, this bit is Reserved
	8	**Edge Indicator [0].** For POLYGON primitive objects, this bit indicates whether the edge from Vertex0 to Vertex1 is an exterior edge of the polygon (i.e., this is the first or only triangle of the polygon). If clear, that edge is an interior edge. The kernel can use this bit to control operations such as generating wireframe representations of polygon primitives. For all other Primitive Topology Types, this bit is Reserved
	7	[**Pre-DevIL**]: Reserved: MBZ **[DevIL+]: Rendering Enabled**: This is a copy of the corresponding GS_STATE/3DSTATE_GS bit . This bit can be used to inform the GS kernel whether or not it needs to output VUEs down the pipeline for possible rendering (as the state which controls whether rendering is enabled can change after the kernel is compiled). Format: U1
	6:5	Reserved

GRF DWord	Bit	Description
	4:0	**Primitive Topology Type.** This field identifies the Primitive Topology Type associated with the primitive containing this object. It indirectly specifies the number of input vertices included in the thread payload. Note that the GS unit may toggle this value between TRISTRIP and TRISTRIP_REV, as described in 4.4.1. Format: See *3D Pipeline*
R0.1	31:0	[DevSNB+]: **Primitive ID**. This field contains the Primitive ID associated with this object. Format: U32
R0.0	31:23	Reserved
	22:16	Reserved
	15:9	Reserved
	8:0	Reserved
Streamed Vertex Buffer Index Values (passed in R1) (only included for [DevCTG+] and SVBI Payload Enable is set.		
R1.7		Maximum Streamed Vertex Buffer Index 3 This is a copy of the **Maximum Index** field sent in the last 3DSTATE_GS_SVB_INDEX command targetting SVBI[3]. The thread will need to use the maximum indices of all bounds SOBs.
R1.6		Maximum Streamed Vertex Buffer Index 2
R1.5		Maximum Streamed Vertex Buffer Index 1
R1.4		Maximum Streamed Vertex Buffer Index 0
R1.3	31:0	Streamed Vertex Buffer Index 3 This field represents the initial value of the index #3. Format = U32 Range = $[0,2^{27}-1]$
R1.2	31:0	Streamed Vertex Buffer Index 2
R1.1	31:0	Streamed Vertex Buffer Index 1
R1.0	31:0	Streamed Vertex Buffer Index 0

GRF DWord	Bit	Description
[Varies] optional	31:0	Constant Data (optional) : **[DevSNB+]:** Some amount of constant data (possible none) can be extracted from the push constant buffer (PCB) and passed to the thread following the R0 Header. The amount of data provided is defined by the sum of the read lengths in the last 3DSTATE_CONSTANT_GS command (taking the buffer enables into account). The Constant Data arrives in a non-interleaved format.
Varies	31:0	**Vertex Data.** There can be up to 6 vertices supplied, each with a size defined by the **Vertex URB Entry Read Length** state. The amount of data provided for each vertex is defined by the **Vertex URB Entry Read Length** state Vertex 0 DWord 0 is located at Rn.0, Vertex 0 DWord 1 is located at Rn.1, etc. Vertex 1 DWord 0 immediately follows the last DWord of Vertex 0, and so on.

4.5 GS Thread Execution

A GS thread is capable of performing arbitrary algorithms given the thread payload (especially vertex) data and associated data structures (binding tables, sampler state, etc.) as input. Output can take the form of vertices output to the FF pipeline (at the GS unit) and/or data written to memory buffers via the DataPort.

The primary usage models for GS threads include (possible combinations of):

- Compiled application-provided "GS shader" programs, specifying an algorithm to convert the vertices of an input object into some output primitives. For example, a GS shader may convert lines of a line strip into polygons representing a corresponding segment of a blade of grass centered on the line. Or it could use adjacency information to detect silhouette edges of triangles and output polygons extruding out from the those edges. Or it could output absolutely nothing, effectively terminating the pipeline at the GS stage.
- Driver-generated instructions used to write pre-clipped vertices into memory buffers (see Stream Output below). This may be required whether or not an app-provided GS shader is enabled.
- Driver-generated instructions used to emulate API functions not supported by specialized hardware. These functions might include (but are not limited to):
    - Conversion of API-defined topologies into topologies that can be rendered (e.g., LINELOOP→LINESTRIP, POLYGON→TRIFAN, QUADs→TRIFAN, etc.)
    - Emulation of "Polygon Fill Mode", where incoming polygons can be converted to points, lines (wireframe), or solid objects.
    - Emulation of wide/sprite points.

[**DevSNB**]: <u>When rendering is required</u>, concurrent GS threads must use the FF_SYNC message (URB shared function) to request an initial VUE handle and synchronize output of VUEs to the pipeline (see *URB* in *Shared Functions*). Only one GS thread can be outputting VUEs to the pipeline at a time. In

order to achieve parallelism, GS threads should perform the GS shader algorithm (along with any other required functions) and buffer results (either in the GRF or scratch memory) before issuing the FF_SYNC message. The issuing GS thread will be stalled on the FF_SYNC writeback until it is that thread's turn to output VUEs. As only one GS thread at a time can output VUEs, the post-FF_SYNC output portion of the kernel should be optimized as much as possible to maximize parallelism.

4.5.1 GS Shader Programming Notes [DevSNB{WA}]

Prior to End of Thread with a URB_WRITE, the kernel must ensure all writes are complete by sending the final write as a committed write.

4.5.2 Vertex Output [DevSNB]

The GS kernel will typically use the URB_WRITE message to output vertices and request additional handles. (Refer to the *3D Pipeline* chapter for a general discussion of how FF units output vertices, and the *URB* chapter for details on the use of the URB_WRITE message.)

The following table lists which primitive topology types are valid for output by a GS thread.

PrimTopologyType	Output
LINELIST	Yes
LINELIST_ADJ	No
LINESTRIP	Yes
LINESTRIP_ADJ	No
LINESTRIP_BF	Yes
LINESTRIP_CONT	Yes
LINESTRIP_CONT_BF	Yes
LINELOOP	No
POINTLIST	Yes
POINTLIST_BF	Yes
POLYGON	Yes
QUADLIST	No
QUADSTRIP	No
RECTLIST	Yes
TRIFAN	Yes

PrimTopologyType	Output
TRIFAN_NOSTIPPLE	Yes
TRILIST	Yes
TRILIST_ADJ	No
TRISTRIP	Yes
TRISTRIP_ADJ	No
TRISTRIP_REV	Yes

The GS thread is responsible for providing correct PrimType, PrimStart and PrimEnd information for each vertex output, in the same fashion as the Vertex Fetch unit. Given that the GS thread is likely performing an algorithm as specified by an application "geometry shader" program, where the algorithm dictates when and if a vertex is to be output, the GS thread is allowed to output incomplete primitives (too few or too many vertices). The downstream FF units will correctly handle any dangling vertices.

However, the PrimStart and PrimEnd indicators must be correct for all vertices, e.g., the last vertex of a topology must have PrimEnd set. This may require the GS thread to postpone completion of a vertex output operation until either the next vertex is encountered or the algorithm (not the thread) completes.

Note that, through use (clearing) of the **Complete** bit in the URB_WRITE message, is it possible to write a vertex to the URB yet delay the "complete" indication until later. The PrimType, PrimStart, and PrimEnd indications are not sampled by the FF pipeline until **Complete** is set. This relieves the GS thread from actually having to buffer the pending vertex.

A GS or CLIP thread is restricted as to the number of URB handles it can retain. Here a "retained" handle refers to a URB handle that (a) has been pre-allocated or allocated and returned to the thread via the **Allocate** bit in the URB_WRITE message, and (b) has yet to be returned to the pipeline via the **Complete** bit in the URB_WRITE message.

- **[DevSNB]**: The number of retained handles must not exceed min(32, **Number of URB Entries**).

This restriction is not expected to be significant in that most/all GS/CLIP threads are expected to retain only a few (<=4) handles.

4.5.3 Stream Output

With a "Stream Output" function, vertex data can be written to one or more memory buffers for subsequent readback by the CPU or use in subsequent Draw operations. The Stream Output function is defined such that the pipeline is tapped immediately following the GS stage (just prior to clipping) and in such a way that permits the GS kernel to perform the writes after the GS shader function.

The final contents of Stream Output buffers must follow the strict pipeline ordering of vertices. Given this ordering requirement, it will be necessary to run the GS stage in a single-threaded fashion (**Maximum Number of Threads** == 1). Otherwise concurrent GS threads might append vertices to the output buffer out of order.

Hardware support for the Stream Output is limited to a special “Streamed Vertex Buffer Write” DataPort message. (Refer to *DataPort* chapter). Through use of this message type, the GS thread can write from 1 to 4 DWords to specified ‘element’ (indexed entry) in a BUFFER surface. The DataPort will inhibit writes past the end of the buffer.

Stream Output is allowed to either a set (<= 4) of “single element buffers” (SEBs) or a single “multiple element buffer” (MEB). The SEB is a simple 1D array of 1-4 DWord elements, while the MEB is a 1D array of structures, with a maximum structure pitch of 2K bytes. Up to 16 1-4 DWord elements within the MEB structure can be written, with arbitrary, multiple-DWord “gaps” that must be left unmodified in memory.

Software will likely need to define separate surface states for each SEB, and separate surface states for each element within the MEB structure. The surfaces are selected via the normal binding table mechanisms.

The need for separate SEB surface states is obvious, as the SEBs are separate buffers in memory. The MEB surface-per-element allows the GS kernel to address the MEB using an structure index. Here each surface would be specified as having the same structure pitch, but with different starting addresses corresponding to the different element offsets within the structure – in effect, defining a set of interleaved surfaces. The GS kernel would output one write message per element.

(Note that software could, if it wished, treat the MEB as a single 1D array of DWords, though it would then have to write the buffer one DWord at a time, performing the address calculations within the GS kernel. This should not be necessary, and is certainly not recommended due to obvious performance and complexity reasons.)

Programming Note: If the GS stage is enabled, software must always allocate at least one GS URB Entry. This is true even if the GS thread never needs to output vertices to the pipeline, e.g., when only performing stream output. This is an artifact of the need to pass the GS thread an initial destination URB handle.

4.5.3.1 Streamed Vertex Buffer Indexing [DevCTG+]

The GS unit supports four Streamed Vertex Buffer Indicies (SVBIs) in hardware. Only when the **Streamed Vertex Buffer Enable** bit (GS_STATE) is set will the current SVBI values be passed to GS threads via R1 of the thread payload. The GS thread is then responsible for (a) using/incrementing these initial values when generating the **Destination Index** field of DataPort Streamed Vertex Buffer Write messages – as the DataPort will this field and not the SVBIs directly to write out vertex data, and (b) correctly programming the Increment SVBIs bit of the DataPort Streamed Vertex Buffer Write message in order to cause the GS’s SVBI values to increment as required. The incremented SVBI values will be passed to the next GS thread unless they are reloaded from the command stream.

The SVBIs can be loaded (either directly or indirectly from memory) via the new 3DSTATE_GS_SVB_INDEX command. Software would use this command to specify initial values when an SVB was bound to the pipeline.

4.5.4 Thread Termination

GS threads must terminate by sending a URB_WRITE message with the **EOT** and **Complete** bits set. The Used bit can be set (if outputting a VUE) or clear (if freeing an used VUE).

4.6 Vertex Header Readback [DevSNB]

The GS unit performs a readback of the Vertex Header of each vertex exiting the GS stage (either passed through or generated by a GS thread) as this information is required by the next FF stage (CLIP). Software is responsible for ensuring that any required Vertex Header fields are valid at this point in the pipeline. See *Vertex Data Overview* for a description of the Vertex Header fields and how they are read-back and used by the GS unit.

4.7 Primitive Output

(This section refers to output from the GS unit to the pipeline, not output from the GS thread)

The GS unit will output primitives (either passed-through or generated by a GS thread) in the proper order. This includes the buffering of a concurrent GS thread's output until the preceding GS thread terminates. Note that the requirement to buffer subsequent GS thread output until the preceding GS thread terminates has ramifications on determining the number of VUEs allocated to the GS unit and the number of concurrent GS threads allowed.

4.8 Other Functionality

4.8.1 Statistics Gathering

There are a number of GS/StreamOutput pipeline statistics counters associated with the GS stage and GS threads. This subsection describes these counters and controls depending on device, even in the cases where functions outside of the GS stage (e.g., DataPort) are involved in the statistics gathering.

Refer to the *Statistics Gathering* summary provided earlier in this specification. Refer to the *Memory Interface Registers* chapter for details on these MMIO pipeline statistics counter registers, as well as the chapters corresponding to the other functions involved (e.g., DataPort, URB shared functions).

4.8.1.1 GS Invocations

The GS unit controls the GS_INVOCATIONS counter, which the number of times a GS thread is executed. A GS thread is executed for each object (triangle, line or point) that is derived from the stream of incoming primitive topologies. If the **Statistics Enable** bit in GS_STATE is set, the GS unit will increment the GS_INVOCATIONS_COUNT register (see Memory Interface Registers in Volume Ia, *GPU*) for each object that is dispatched to a GS thread.

4.8.1.2 GS Primitives Output [DevSNB]

The GS_PRIMITIVES_COUNT pipeline statistics register counts objects (triangles/lines/points) output by GS threads.

4.8.1.2.1 GS Primitives Output [DevSNB]

As a effect of GS threads issuing FF_SYNC messages to the URB shared function, the GS_PRIMITIVES_COUNT register is incremented by the **NumGSPrimsGenerated** field of that message.

4.8.1.3 Stream Output Primitives Written [DevSNB]

GS threads must terminate by issuing a URBWrite message with EOT set. The URBWrite header contains an SONumPrimsWritten Increment Count

Whenever a GS thread outputs a DataPort Streamed Vertex Buffer Write (SVBWrite) message with the **Increment Num Prims Written** bit set, the SO_NUM_PRIMS_WRITTEN register will be incremented. The **Statistics Enable** bit in GS_STATE does not affect the increment of this register. [**DevGT+**]: The **SO Statistics Enable** bit (GS_STATE) controls whether the SO_NUM_PRIMS_WRITTEN register is incremented.

Programming Note: The GS thread is solely responsible for limiting the increment of SO_NUM_PRIMS_WRITTEN in the face of SVB buffer overflow. There is no hardware performing this function.

4.8.1.4 Stream Output Primitive Storage Needed [DevSNB]

Whenever a GS thread outputs a DataPort Streamed Vertex Buffer Write (SVBWrite) message with the **Increment Prim Storage Needed** bit set, the SO_NUM_PRIM_STORAGE_NEEDED register will be incremented. The **Statistics Enable** bit in GS_STATE does not affect the increment of this register. [**DevSNB+**]: The **SO Statistics Enable** bit (GS_STATE) controls whether the SO_PRIM_STORAGE_NEEDED register is incremented.

[**DevSNB+]:** In addition to (and, frankly, in lieu of) the control mentioned above, DevSNB+ adds an additional method of incrementing this counter. As a effect of GS threads issuing FF_SYNC messages to the URB shared function, the SO_PRIM_STORAGE_NEEDED register is incremented by the **NumSOPrimsNeeded** field of that message. Note that this new method removes any need for the GS thread to issue (possibly multiple) dummy SVBWrite messages simply to increment SO_PRIM_STORAGE_NEEDED.

Programming Note: There should be no need for GS threads to limit the increment of SO_PRIM_STORAGE_NEEDED, as this value should reflect the minimum buffer size required to avoid overflow.

5. Clip Stage

5.1 CLIP Stage Overview

The CLIP stage of the 3D Pipeline is similar to the GS stage in that it can be used to perform general processing on incoming 3D objects via spawned threads. However, the CLIP stage also includes specialized logic to perform a *ClipTest* function on incoming objects. These two usage models of the CLIP stage are outlined below.

Refer to the *Common 3D FF Unit Functions* subsection in the *3D Overview* chapter for a general description of a 3D Pipeline stage, as much of the CLIP stage operation and control falls under these "common" functions. I.e., many of the CLIP stage state variables and CLIP thread payload parameters are described in *3D Overview*, and although they are listed here for completeness, that chapter provides the detailed description of the associated functions.

Refer to this chapter for an overall description of the CLIP stage, details on the ClipTest function, and any exceptions the CLIP stage exhibits with respect to common FF unit functions.

5.1.1 Clip Stage – General-Purpose Processing

Numerous state variable controls are provided to tailor the ClipTest function as required by the API or primitive characteristics. These controls allow a mode where all objects are passed to CLIP threads, and in this regard the CLIP stage can be used as a second GS stage. However, unlike the GS stage, primitives output by CLIP threads will not be subject to 3D Clipping, and therefore any clip-testing/clipping of these primitives (if required) would need to be performed by the CLIP thread itself.

5.1.2 Clip Stage – 3D Clipping

The ClipTest fixed function is provided to optimize the CLIP stage for support of generalized *3D Clipping*. The CLIP FF unit examines the position of incoming vertices, performs a fixed function *VertexClipTest* on these positions, and then examines the results for the vertices of each independent object in *ClipDetermination*.

The results of ClipDetermination indicate whether an object is to be processed by a thread (MustClip), discarded (TrivialReject) or passed down the pipeline unmodified (TrivialAccept). In the MustClip case, the spawned thread is responsible for performing the actual 3D Clipping algorithm. The CLIP thread is passed the source object vertex data and is able to output a new, arbitrary 3D primitive (e.g., the clipped primitive), or no output at all. Note that the output primitive is independent in that it is comprised of newly-generated VUEs, and does not share vertices with the source primitive or other CLIP-generated primitives.

New vertices produced by the CLIP threads are stored in the URB. Their Vertex Headers are then read from the VUEs in order to insert the relevant information into the 3D pipeline. The CLIP unit maintains the proper ordering of CLIP-generated primitives and any surrounding trivially-accepted primitives. The CLIP unit also supports multiple concurrent CLIP threads and maintains the proper ordering of the thread outputs as dictated by the order of the source objects.

The outgoing primitive stream is sent down the pipeline to the Strip/Fan (SF) FF stage (now including the read-back VUE Vertex Header data such as Vertex Rosition (NDC or screen space), RTAIndex, VPIndex, PointWidth) and control information (PrimType, PrimStart, PrimEnd) while the remainder of the vertex data remains in the VUE in the URB.

5.1.3 [DevSNB+] Fixed Function Clipper

[**DevSNB+**] The device supports Fixed Function Clippling. Prior to this fixed function pipeline had Clipping done in the EU. However the clipper thread latency was high and caused a bottleneck in the pipeline. Hence the motivation for a fixed function clipper.

5.2 Concepts

This section provides an overview of 3D clip-testing and clipping concepts, as defined by the D3D and OpenGL APIs. It is provided as background material: some of the concepts impact HW functionality while others impact CLIP kernel functionality.

5.2.1 The Clip Volume

3D objects are optionally clipped to the *clip volume*. The clip volume is defined as the intersection of a set of *clip half-spaces*. Six of these half-spaces define the view volume, while additional, user-defined half-spaces can be employed to perform clipping (or at least culling) within the view volume.

The CLIP stage design will permit the enable/disable of certain subsets of these clip half-spaces. This capability can be used, for example, to disable viewport, guardband, and near and far clipping as required by the API and other conditions.

5.2.1.1 View Volume

The intersection of the six view half-spaces defines the *view volume*. The view volume is defined in 4D clip space coordinates as:

View Clip Plane	'Outside' Condition	
	4D Clip Space	NDC space, positive w
XMIN (NDC Left)	clip.x < -clip.w	ndc.x < -1
XMAX (NDC Right)	clip.w < clip.x	ndc.x > 1
YMIN (NDC Bottom)	clip.y < -clip.w	ndc.y < -1
YMAX (NDC top)	clip.w < clip.y	ndc.y > 1

View Clip Plane	'Outside' Condition	
	4D Clip Space	NDC space, positive w
ZMIN (NDC Near)	D3D: clip.z < 0.0 OGL: clip.z < -clip.w	D3D: ndc.z < 0.0 OGL: ndc.z < -1.0
ZMAX (NDC Far)	clip.w < clip.z	ndc.z > 1.0

Note that, since the 2D (X,Y) extent of the projected view volume is subsequently mapped to the 2D pixel space viewport, the terms "viewport" and "view volume" are used somewhat interchangeably in this discussion.

The CLIP unit will perform view volume clip test using NDC coordinates (the results of the speculative PerspectiveDivide). The treatment of negative ndc.w and invalid (NaN, +/-INF) coordinates is clarified below.

Negative W Coordinates

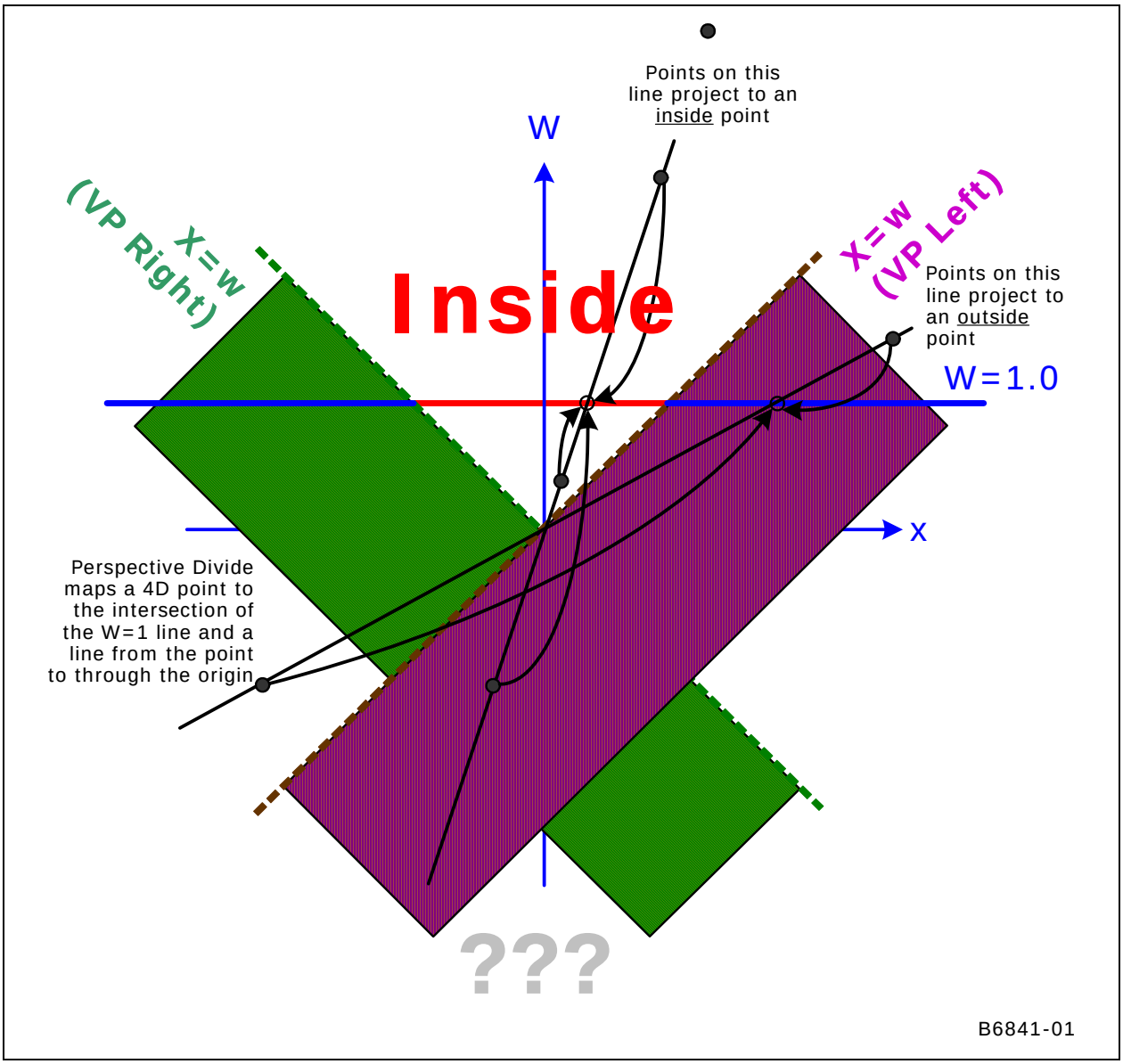
Consider for a moment vertices with a negative clip.w coordinate. Examination of the API definitions for "outside" shows that it is impossible for that vertex to be considered inside both the XMIN (NDC Left) and XMAX (NDC Right) planes. The clip.x coordinate would need to be greater than or equal to some positive value (-clip.w) to be considered inside the XMIN plane, while also being less than or equal to the negative (clip.w) value to be considered inside the XMAX plane. Obviously both these conditions cannot be met simultaneously, so a vertex with a negative clip.w coordinate will always appear outside.

Surprisingly, it is possible for a vertex to be outside both the XMIN and XMAX planes (and likewise for the Y axis). This arises when clip.w is negative and clip.x falls between clip.w and -clip.w. Note, however, that in NDC space (post perspective-divide), this same vertex would be considered inside. This disparity arises from the loss of information from the perspective divide operation, specifically the signs of the input operands. The CLIP stage will avoid this artifact by supporting an additional clip.w=0 clip plane – a negative ndc.rhw value indicates the point is outside of the clip.w=0 plane. (See sections below for related errata in DevBW and DevCL devices)

The assumption made in the Clip stage is that only the w>0 portion of clip space is considered visible. The VertexClipTest function tests each incoming 1/w value and, if negative, the vertex is tagged as being outside the w=0 plane. These vertex outcodes are combined in ClipDetermination to determine TA/TR/MC status.

A negative w coordinate poses an additional issue due to the fact that VertexClipTest is performed using post-perspection-projection coordinates (NDC or screen space). This disparity arises from the loss of information from the perspective divide operation, specifically the signs of the input operands. For example, to test for (x>w) using NDC coordinates, (x/w>1) must be used when w>0, and (x/w<1) must be used when w<0. The VertexClipTest function therefore uses the sign of the incoming 1/w coordinate to select the appropriate comparison function for each of the VP and GB clip planes.

As the CLIP thread performs clipping in 4D clip space, only the truly visible portions of objects (i.e, meeting the 4D clip space visibility criteria) will be considered. The CLIP thread should not output negative w (clip or NDC) coordinates.



5.2.2 User-Specified Clipping

The various APIs define mechanisms by which objects can be clipped or culled according to some user-specified parameter(s) in addition to the implied viewport clipping. The HW support of these mechanisms is restricted to use of the 8 UserClipFlags (UCFs) of the VUE Vertex Header. Software is required to provide the remaining support (e.g., the JITTER including instructions to cause a distance value to be computed, tested for visibility, and generation of the appropriate UCF bit.)

5.2.2.1 User Clip Planes (OGL)

In OpenGL, up to 6 *user clip planes* can be defined and enabled. These planes define half-spaces that are intersected with the view volume (and each other) to form a final clip volume. Each user clip plane is specified by four coefficients of a plane equation in clip space coordinates (`UserClipPlane[n].xyzw`). A point is not visible if it has a negative distance to the plane. Therefore, points P that satisfy the following equation are considered to lie in the half-space and therefore may be visible:

(P.xyzw dot UCP[n].xyzw) >= 0, 0<=n<=5

There is no direct HW support for this distance computation. The driver/JITTER is required to cause the distances to be correctly computed/compared in a shader, with the comparison result (boolean) placed in the proper location in the Vertex Header.

5.2.3 Negative-W Clipping Errata

In legacy devices, there is a bug in the definition of the handling of negative RHW (1/w) coordinates in the Clip unit's trivial reject logic. The fault may cause line and triangle objects to be erroneously trivially rejected and therefore be manifested as occasional missing geometry.

This section defines a correction of the problem in DevCTG+.

A new "NEGW" vertex outcode is added. It is set for a vertex if the RHW component is negative. Also invert the computed VP,GB vertex outcodes if NEGW is seen. In ClipDetermination, NEGW is treated like a separate clip plane in determination of trivial accept, trivial reject and mustclip cases.

Table 13 SW Workaround Summary

Device	VS/GS Kernel	Clip State	Clip Kernel	Notes
DevCTG+	No WA required	None, other than enabling NEGW clip	None.	

5.2.4 Guard Band

***Note**: Refer to Vertex X,Y Clamping and Quantization in the SF stage section for device-specific guardband size information.*

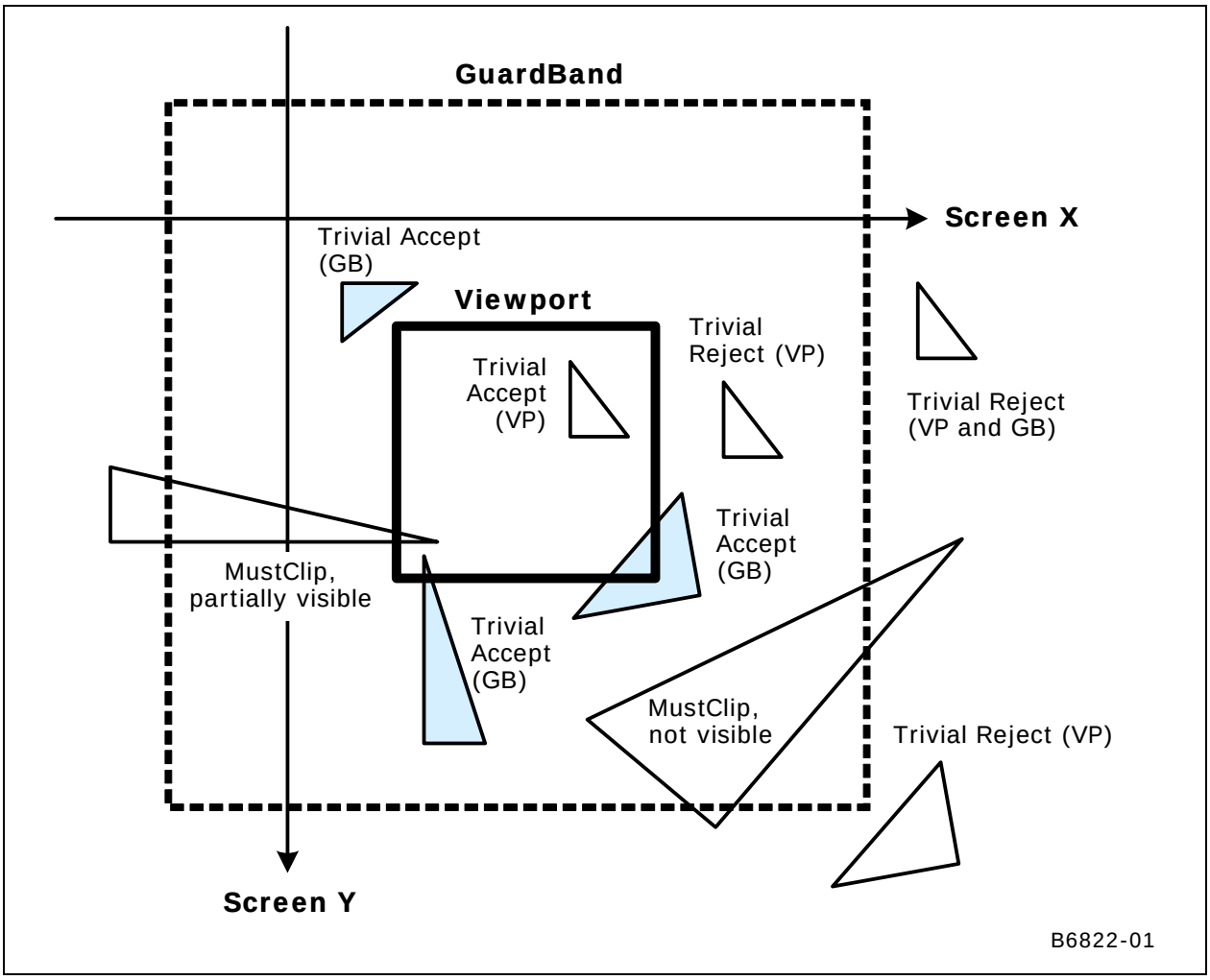
3DClipping is time consuming. For cases where 2DClipping is sufficient, we are willing to forgo 3DClipping and instead apply 2DClipping during rendering. In the general case, this is possible only when an object is totally within the ZMin and ZMax planes, and only clipping to the view volume X/Y MIN/MAX clip planes is required, as 2DClipping is restricted to a screen-aligned 2D rectangle.

However, we must ensure that the 2D extent of these objects do not exceed the limitations of the renderer's coordinate space (see Vertex X,Y Clamping and Quantization in the SF section). Therefore we define a 2D *guardband* region corresponding to (though likely somewhat smaller than) the maximum 2D extent supported by the renderer. During VertexClipTest, vertices are (optionally) subjected to an additional visibility test based on the 2D guardband region.

During ClipDetermination, if an object is not trivially-rejected from the 2D viewport, the XMIN_GB, XMAX_GB, YMIN_GB and YMAX_GB guardband outcodes are used instead of the XMIN, XMAX, YMIN, YMAX view volume outcodes to determine trivial-accept. This will allow objects that fall within the guardband and possibly intersect the viewport to be trivially-accepted and passed down the pipeline.

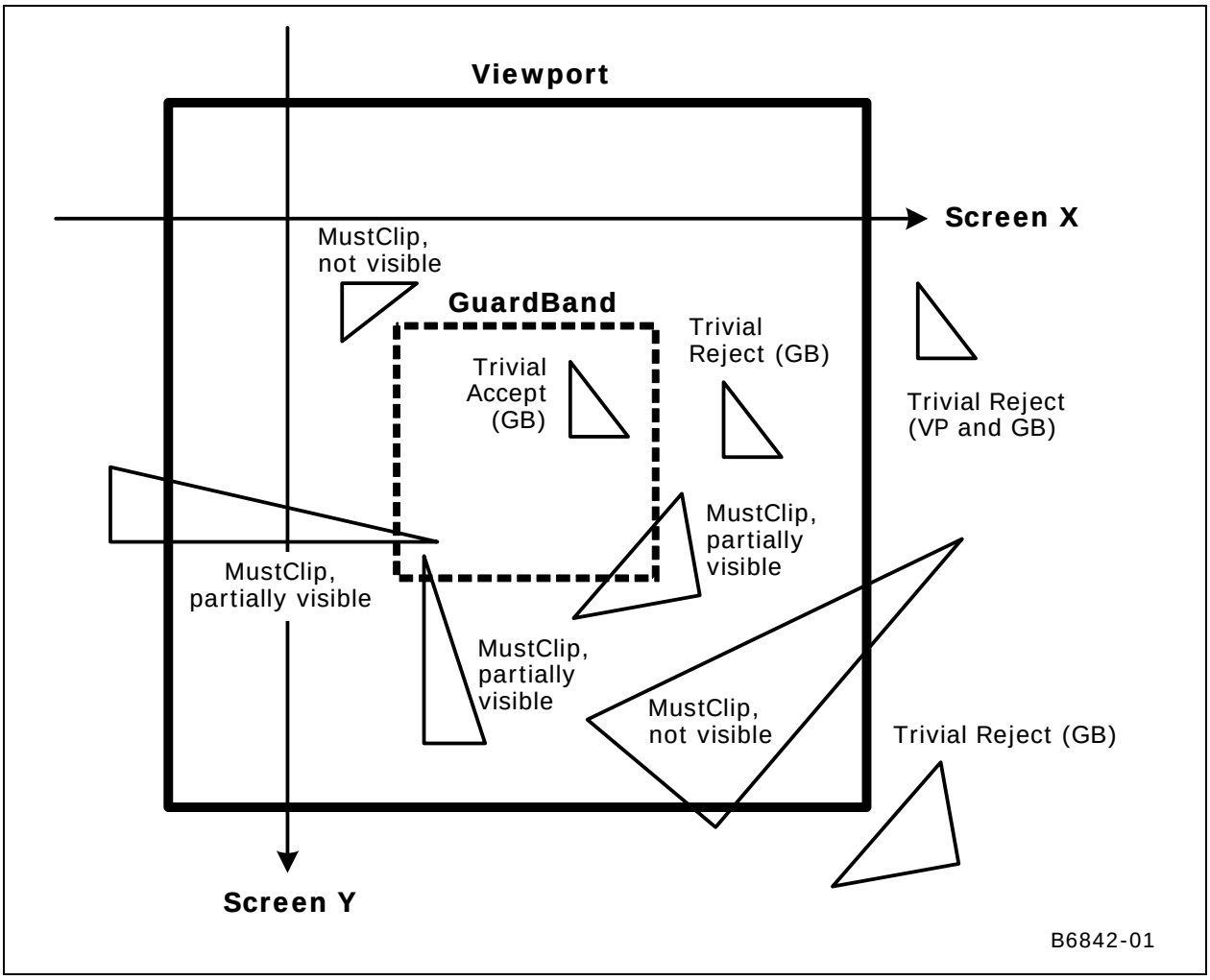
The diagram below shows some examples of objects (triangles) in relation to the viewport and guardband. The shaded triangles are examples of triangles that are not trivially accepted to the viewport but trivially accepted to the guardband and therefore passed to down the pipeline. Without the guardband, these triangles would have to be submitted to a CLIP thread.

Figure 5-1. Normal Guardband Operation



The CLIP stage needs to handle the case where the viewport XY is larger than the screen space coordinate range supported by the SF and WM units. This condition may arise when the API defines an implicit 2D clip between the viewport XY extent and the rendertarget. In the 3D pipeline, the guardband must used to force explicit clipping in order to ensure legal coordinates are passed out of the CLIP stage. Therefore the CLIP unit supports a guardband that can be larger or smaller than the viewport (in any particular direction). The following diagram illustrates a case with a very large viewport, extending well beyond the guardband. Note that the only trivial accept case is where objects are completely within the guardband.

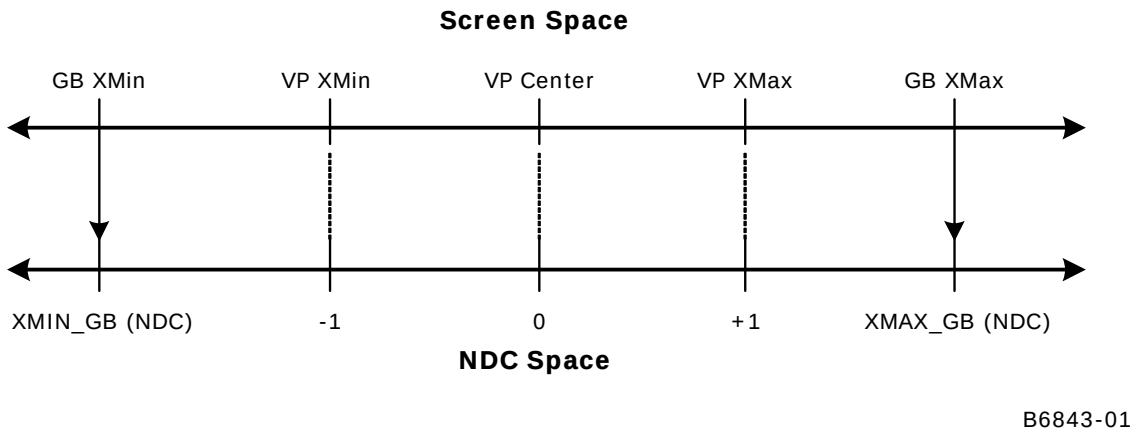
Figure 5-2. Very Large Viewport Case



5.2.4.1 NDC Guardband Parameters

Note: *Refer to Vertex X,Y Clamping and Quantization in the SF stage section for device-specific guardband size information.*

When the CLIP unit performs VertexClipTest in NDC space, the guardband limits must be provided as NDC coordinates. The diagram below shows how the guardband NDC coordinates are derived. Specifically, the XMIN_GB NDC coordinate is simply the ratio of the (screen space) distance from the screen space VP center to the screen space GB XMin boundary over the distance from the VP center to the VP XMin (left) boundary. A similar computation yields the XMAX_GB (right), YMIN_GB (bottom) and YMAX_GB (top) guardband NDC coordinates.



As these guardband parameters are defined relative to the viewport, each of the up-to-16 sets of viewport specifications supported in the 3D pipeline will require a corresponding set of guardband parameters. These guardband parameters are provided as a separate memory-resident state structure (CLIP_VIEWPORT), and referenced via the **Clipper Viewport State Pointer** contained in the CLIP_STATE structure. Note that the CLIP_VIEWPORT structure has a different definition than the SF_VIEWPORT structure used by the SF unit.

5.2.5 Vertex-Based Clip Testing and Considerations

The CLIP unit performs clip test and determines whether objects need to be clipped based solely on information (position, UserClipFlags) provided at the vertices of the object as they arrive at the clip stage. Issues arise if and when the corresponding rendered object is not constrained to the convex hull of the object. Different APIs impose different treatment of these conditions.

In addition and in the more general case, a CLIP thread could be used to convert the object (as defined by its vertices) into some arbitrary output primitive. In this case, the CLIP unit's ClipTest/ClipDetermination logic may not be suitable for determination of when to reject/accept/clip objects. In this case the ClipMode can be used to route all (or all non-rejected) objects to CLIP threads, where the proper clip-test and clipping can occur in the CLIP kernel.

One issue that arises is whether a trivial-reject to the VPXY is suitable. If this were allowed, an object might be discarded even if it would have been partially visible in the viewport. A second issue is whether a TA against the GB is suitable. If this were allowed, portions of the rendered object might be visible in the VP even if the object should have been clipped out of the VP.

5.2.5.1 Triangle Objects

In the normal processing of triangle-based primitives (tristrip/trilist/polygon/etc.), the footprint of each triangle is constrained to the 2D convex hull. I.e., the rendering of these triangles will not produce pixels outside of the triangle. Therefore the normal operation of the CLIP unit functions will support the proper clip testing and clip determination for triangle objects:

- Both the VPXY and GB clip boundaries can be utilized (as described above). If the triangle is TR against the VP, it can be discarded. Otherwise, if the triangle is TA against the GB, it can be passed down the pipeline (assuming it is TA against VPZ, UCFs, etc.) and properly handled by 2DClipping.

- The GB parameters can be programmed to coincide with the maximum allowable screen space extent (though making the GB marginally smaller than this max extent is highly recommended).

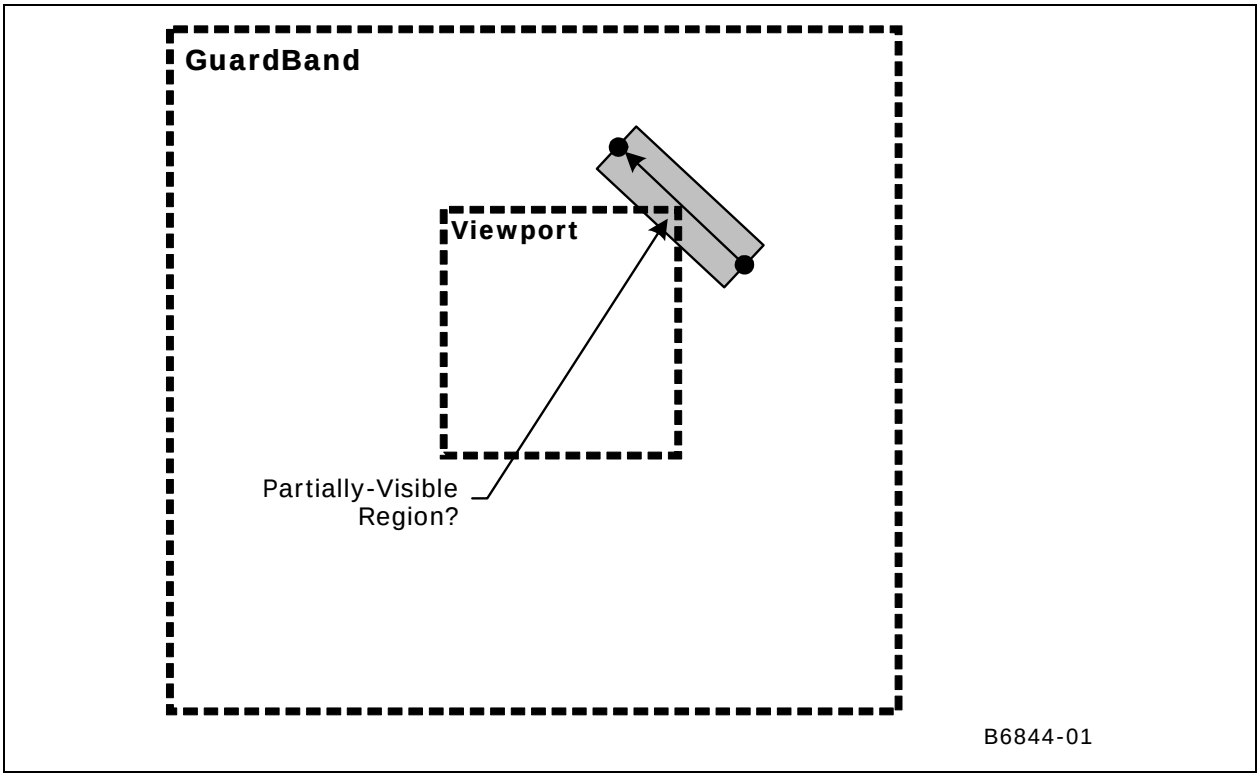
5.2.5.2 Non-Wide Line Objects

In the normal processing of non-wide, line-based primitives (linestrip/linelist/etc.), the footprint of each line is constrained to the 2D convex hull. I.e., the rendering of these lines will not produce pixels off of the line. Therefore the normal operation of the CLIP unit functions will support the proper clip testing and clip determination for non-wide line objects. (See Triangle Objects above).

5.2.5.3 Wide Line Objects

The rendering hardware supports wide lines (solid lines with a line width or anti-aliased lines). When rendered, pixels outside of the convex hull will be generated.

The following diagram shows an example of a wide line that normally would be TA against the GB. If the TA is allowed, the partially-visible region of the line would be rendered.

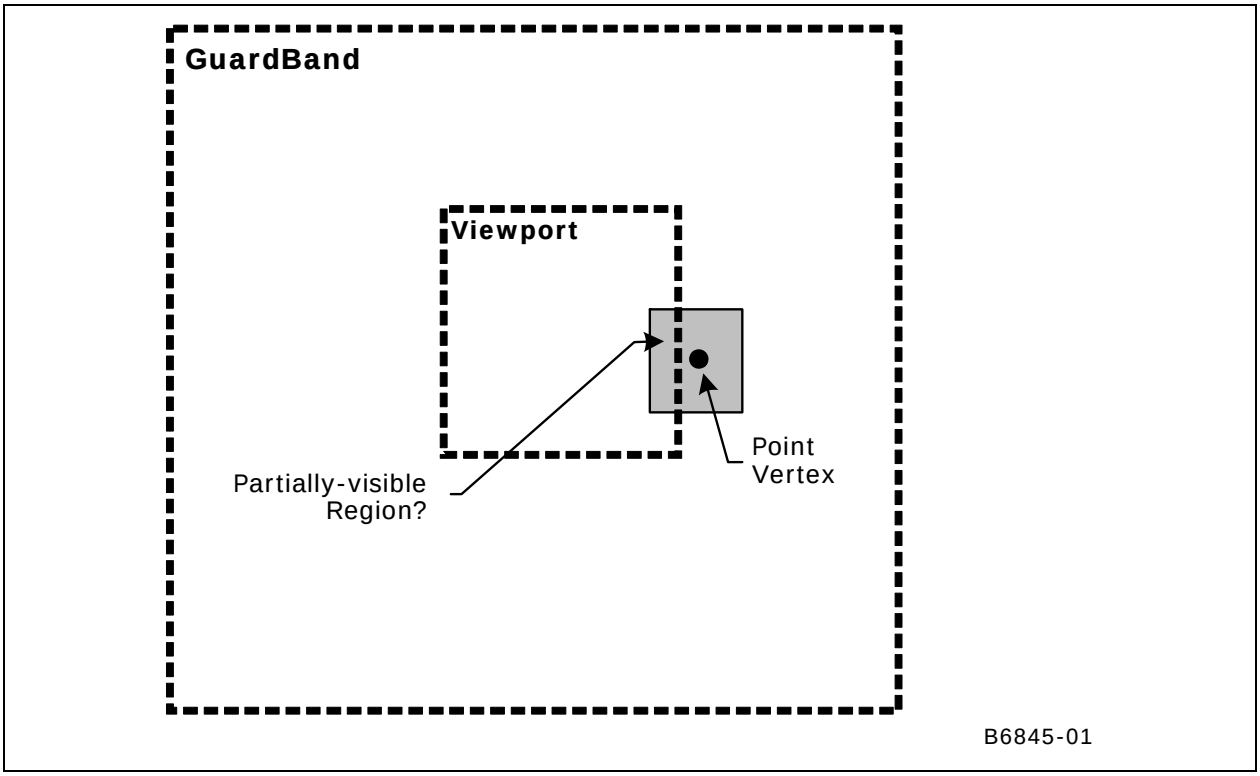


In general, OpenGL dictates that the partially-visible region must not be rendered. In this case the line must be clipped-out against the VPXY (not TA against the GB). To accomplish this, SW could disable the GB when drawing wide lines.

5.2.5.4 Wide Points

The rendering hardware supports a width parameter for native line objects. When rendered, pixels surrounding the point (center) vertex will be generated.

The following diagram shows an example wide point that normally would be TR against the VPXY. If the TR is allowed, the partially-visible region of the point would not be rendered.



In general, OpenGL dictates that the partially-visible region must not be rendered. In this case the point must be TR against the VPXY (not TA against the GB). To accomplish this, SW could disable the GB when drawing wide points.

In D3D, the partially visible region should be rendered. (This behavior reduces 'popping' artifacts as the point center is perturbed about the VP boundaries.) To accomplish this, software could disable the VPXY and leave the GB enabled. In this case, software might want to set the GB to extend partially past the VPXY (to filter out points that cannot be visible, but otherwise would be TR against a large GB).

5.2.5.5 RECTLIST

The CLIP unit treats RECTLIST exactly like TRILIST. No special consideration is made for the implied 4th vertex of each rectangle (although ViewportXY and Guardband VertexClipTest theoretically should be sufficient to drive ClipDetermination). Given this, and the fact that RECTLIST is primarily intended for driver-generated "BLT" functions, there are number of restrictions on the use of RECTLIST, especially regarding the CLIP unit. Refer to the RECTLIST definition in 3D Pipeline.

5.2.6 3D Clipping

If an object needs to be clipped, it will be passed to the CLIP thread. The CLIP thread will perform some (arbitrary) algorithm to clip the primitive, and subsequently output "new" vertices as a primitive defining the visible region of the input object (assuming there is a visible region). In the process of spawning the CLIP thread, the input vertices may be considered "consumed" and therefore dereferenced. Therefore the CLIP thread will need to copy (if required) any input VUE data to a new output VUE – there is no mechanism to "output" input vertices other than copying.

Note: [DevSNB+] supports only Fixed function Clipping

5.3 CLIP Stage Input

As a stage of the 3D pipeline, the CLIP stage receives inputs from the previous (GS) stage. Refer to *3D Overview* for an overview of the various types of input to a 3D Pipeline stage. The remainder of this subsection describes the inputs specific to the CLIP stage.

5.3.1 State

5.3.1.1 3DSTATE_CLIP [DevSNB]

For **[DevSNB]**, the state used by the clip stage is defined with this inline state packet.

3DSTATE_CLIP			
Project:	[DevSNB]	**Length Bias:**	2

DWord	Bit	Description
0	31:29	Command Type Default Value: 3h GFXPIPE Format: OpCode
	28:27	Command SubType Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	3D Command Opcode Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	3D Command Sub Opcode Default Value: 12h 3DSTATE_ Format: OpCode
	15:8	Reserved Project: All Format: MBZ
	7:0	DWord Length Default Value: 02h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All

3DSTATE_CLIP

1	31:11	Reserved Project: All Format: MBZ
	10	Clipper Statistics Enable Project: All Format: Enable This bit controls whether Clip-unit-specific statistics register(s) can be incremented. Value / Name / Description / Project 0h / Disable / CL_INVOCATIONS_COUNT cannot increment / All 1h / Enable / CL_INVOCATIONS_COUNT can increment / All
	9:8	Reserved Project: All Format: MBZ
	7:0	User Clip Distance Cull Test Enable Bitmask Project: All Format: 8-bit mask FormatDesc This 8 bit mask field selects which of the 8 user clip distances against which trivial reject / trivial accept determination needs to be made (does not cause a must clip). DX10 allows simultaneous use of ClipDistance and Cull Distance test of up to 8 distances. Clip Distance Cull Test Enable Bitmask" and "Clip Distance Clip Test Enable Bitmask" should not have overlapping bits in the mask, else the results are undefined.
2	31	CLIP Enable Project: All Format: Enable FormatDesc Specifies whether the CLIP function is enabled or disabled (pass-through).
	30	API Mode Project: All Controls the definition of the NEAR clipping plane Value / Name / Description / Project 0h / APIMODE_OGL / NEAR VP boundary == 0.0 (NDC) / All 1h / APIMODE_D3D / NEAR VP boundary == -1.0 (NDC) / All
	29	Reserved Project: All Format: MBZ
	28	Viewport XY ClipTest Enable Project: All Format: Enable This field is used to control whether the Viewport X,Y extents are considered in VertexClipTest. See Tristrip Clipping Errata subsection.

3DSTATE_CLIP		
	27	Viewport Z ClipTest Enable Project: All Format: Enable This field is used to control whether the Viewport Z extents (near, far) are considered in VertexClipTest.
	26	Guardband ClipTest Enable Project: All Format: Enable This field is used to control whether the Guardband X,Y extents are considered in VertexClipTest for non-point objects. If the Guardband ClipTest is DISABLED but the Viewport XY ClipTest is ENABLED, ClipDetermination operates as if the Guardband were coincident with the Viewport. If both the Guardband and Viewport XY ClipTest are DISABLED, all vertices are considered “visible” with respect to the XY directions.
	25	Reserved Project: All Format: Enable
	24	Reserved Project: All Format: MBZ
	23:16	User Clip Distance Clip Test Enable Bitmask Project: All Format: 8-bit mask FormatDesc This 8 bit mask field selects which of the 8 user clip distances against which trivial reject / trivial accept / must clip determination needs to be made. DX10 allows simultaneous use of ClipDistance and Cull Distance test of up to 8 distances. Clip Distance Cull Test Enable Bitmask" and "Clip Distance Clip Test Enable Bitmask" should not have overlapping bits in the mask, else the results are undefined.

3DSTATE_CLIP

15:13 — Clip Mode

Project: All

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

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

12:10 — Reserved Project: All Format: MBZ

9 — Perspective Divide Disable

Project: All

Format: Disable

This field disables the Perspective Divide function performed on homogeneous position read from the URB. This feature can be used by software to submit pre-transformed "screen-space" geometry for rasterization. This likely requires the W component of positions to contain "rhw" (aka 1/w) in order to support perspective-correct interpolation of vertex attributes. Likewise, the X,Y,Z components will likely be required to be X/W, Y/W, Z/W.

Note that the device does not support clipping when perspective divide is disabled. Software must specify CLIPMODE_ACCEPT_ALL whenever it disables perspective divide. This implies that software must ensure that object positions are completely contained within the "guardband" screen-space limits imposed by the SF unit (e.g., by clipping in CPU SW before submitting the objects).

[errata] Clipper might use the incorrect version of this field when processing back to back primitives with different values of this bit. When changing the value of this bit, an extra 3DSTATE_CLIP followed by a PIPE_CONTROL needs to be inserted before the new 3DSTATE_CLIP.

3DSTATE_CLIP

	8	Non-Perspective Barycentric Enable Project: All Format: Enable FormatDesc This field enables computation of non-perspective barycentric parameters in the clipper, which are sent to SF unit in the must clip case. This field must be enabled if any non-perspective barycentric parameters are enabled in the Windower.
	7:6	Reserved Project: All Format: MBZ
	5:4	Triangle Strip/List Provoking Vertex Select Project: All Format: U2 enumerated type FormatDesc This field selects which vertex of a triangle (in a triangle strip or list primitive) is considered the “provoking vertex”.

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

	3:2	Line Strip/List Provoking Vertex Select Project: All Format: U2 enumerated type FormatDesc This field selects which vertex of a line (in a line strip or list primitive) is considered the “provoking vertex”.

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

	1:0	Triangle Fan Provoking Vertex Select Project: All Format: U2 enumerated type FormatDesc This field selects which vertex of a triangle (in a triangle fan primitive) is considered the “provoking vertex”.

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

3DSTATE_CLIP		
3	31:28	Reserved Project: All Format: MBZ
	27:17	Minimum Point Width Project: All Format: U8.3 pixels This value is used to clamp read-back PointWidth values.
	16:6	Maximum Point Width Project: All Format: U8.3 pixels This value is used to clamp read-back PointWidth values.
	5	Force Zero RTAIndex Enable Project: All Format: Enable If set, the Clip unit will ignore the read-back RTAIndex and operate as if the value 0 was read-back. If clear, the read-back value is used.
	4	Reserved Project: All Format: MBZ
	3:0	Maximum VPIndex Project: All Format: U4 index value (# of viewports -1) This field specifies the maximum valid VPIndex value, corresponding to the number of active viewports. If the source of the VPIndex exceeds this maximum value, a VPIndex value of 0 is passed down the pipeline. Note that this clamping does not affect a VPIndex value stored in the URB.

5.3.1.2 CLIP_VIEWPORT [DevSNB]

[DevSNB]: The viewport-related state is stored as an array of up to 16 elements, each of which contains the DWords described here. The start of each element is spaced 4 DWords apart. The first element of the viewport state array is aligned to a 32-byte boundary, and is located at (**General State Base Pointer + Clipper Viewport State Pointer)**. Note that the definition of the CLIP_VIEWPORT structure differs from the SF_VIEWPORT structure used by the SF unit.

CLIP_VIEWPORT			
Project:	DevSNB	**Length Bias:**	2
Viewport data used by the Clip unit.			
DWord	**Bit**	**Description**	
0	31:0	XMin Clip Guardband Project: All Format: FLOAT32 FormatDesc For VPOS_NDCSPACE: This 32-bit float represents the XMin guardband boundary (normalized to Viewport.XMin == -1.0f). This corresponds to the left boundary of the NDC guardband. For: VPOS_SCREENSPACE This 32-bit float represents the XMin guardband boundary in screen space coordinates. This corresponds to the left boundary of the screen space guardband.	
1	31:0	XMax Clip Guardband Project: All Format: FLOAT32 FormatDesc For VPOS_NDCSPACE: This 32-bit float represents the XMax guardband boundary (normalized to Viewport.XMax == 1.0f). This corresponds to the right boundary of the NDC guardband. For: VPOS_SCREENSPACE This 32-bit float represents the XMax guardband boundary in screen space coordinates. This corresponds to the right boundary of the screen space guardband.	
2	31:0	YMin Clip Guardband Project: All Format: FLOAT32 FormatDesc For VPOS_NDCSPACE: This 32-bit float represents the YMin guardband boundary (normalized to Viewport.YMin == -1.0f). This corresponds to the bottom boundary of the NDC guardband. For: VPOS_SCREENSPACE This 32-bit float represents the YMin guardband boundary in screen space coordinates. This corresponds to the top boundary of the screen space guardband.	

CLIP_VIEWPORT		
3	31:0	YMax Clip Guardband Project: All Format: FLOAT32 FormatDesc For VPOS_NDCSPACE: This 32-bit float represents the YMax guardband boundary (normalized to Viewport.YMax == 1.0f). This corresponds to the top boundary of the NDC guardband. For: VPOS_SCREENSPACE This 32-bit float represents the YMax guardband boundary in screen space coordinates. This corresponds to the bottom boundary of the screen space guardband.

5.3.1.3 CLIP_VIEWPORT

The viewport-related state is stored as an array of up to 16 elements, each of which contains the DWords described here. The start of each element is spaced 4 DWords apart. The first element of the viewport state array is aligned to a 32-byte boundary, and is located at (**General State Base Pointer** + **Clipper Viewport State Pointer)**.

Note that the definition of the CLIP_VIEWPORT structure differs from the SF_VIEWPORT structure used by the SF unit.

CLIP_VIEWPORT		
Project: All		
Viewport data used by the Clip unit.		
DWord	**Bit**	**Description**
0	31:0	XMin Clip Guardband Project: All Format: FLOAT32 For VPOS_NDCSPACE: This 32-bit float represents the XMin guardband boundary (normalized to Viewport.XMin == -1.0f). This corresponds to the left boundary of the NDC guardband. For: VPOS_SCREENSPACE This 32-bit float represents the XMin guardband boundary in screen space coordinates. This corresponds to the left boundary of the screen space guardband.
1	31:0	XMax Clip Guardband Project: All Format: FLOAT32 For VPOS_NDCSPACE: This 32-bit float represents the XMax guardband boundary (normalized to Viewport.XMax == 1.0f). This corresponds to the right boundary of the NDC guardband. For: VPOS_SCREENSPACE This 32-bit float represents the XMax guardband boundary in screen space coordinates. This corresponds to the right boundary of the screen space guardband.

CLIP_VIEWPORT		
2	31:0	YMin Clip Guardband Project: All Format: FLOAT32 For VPOS_NDCSPACE: This 32-bit float represents the YMin guardband boundary (normalized to Viewport.YMin == -1.0f). This corresponds to the bottom boundary of the NDC guardband. For: VPOS_SCREENSPACE This 32-bit float represents the YMin guardband boundary in screen space coordinates. This corresponds to the top boundary of the screen space guardband.
3	31:0	YMax Clip Guardband Project: All Format: FLOAT32 For VPOS_NDCSPACE: This 32-bit float represents the YMax guardband boundary (normalized to Viewport.YMax == 1.0f). This corresponds to the top boundary of the NDC guardband. For: VPOS_SCREENSPACE This 32-bit float represents the YMax guardband boundary in screen space coordinates. This corresponds to the bottom boundary of the screen space guardband.

5.4 VertexClipTest Function

The VertexClipTest function compares each incoming vertex position (x,y,z,w) with various viewport and guardband parameters (either hard-coded values or specified by state variables).

The RHW component of the incoming vertex position is tested for NaN value, and if a NaN is detected, the vertex is marked as “BAD” by setting the outcode[BAD]. If a NaN is deteced in any vertex homogeneous x,y,z,w component or an enabled ClipDistance value, the vertex is marked as “BAD” by setting the outcode[BAD].

In general, any object containing a BAD vertex will be discarded, as how to clip/render such objects is undefined. However, in the case of CLIP_ALL mode, a CLIP thread will be spawned even for objects with “BAD” vertices. The CLIP kernel is required to handle this case, and can examine the **Object Outcode [BAD]** payload bit to detect the condition. (Note that the VP and GB Object Outcodes are UNDEFINED when BAD is set).

If the incoming RHW coordinate is negative (including negative 0) the NEGW outcode is set. Also, this condition is used to select the proper comparison functions for the VP and GB outcode tests (below).

Next, the VPXY and GB outcodes are computed, depending on the corresponding enable SV bits. If one of VPXY or GB is disabled, the enabled set of outcodes are copied to the disabled set of outcodes. This effectively defines the disabled boundaries to coincide with the enabled boundaries (i.e., disabling the GB is just like setting it to the VPXY values, and vice versa.).

The VPZ outcode is computed as required by the API mode SV.

Finally, the incoming UserClipFlags are masked and copied to corresponding outcodes.

The following algorithm is used by VertexClipTest:

```
//
// Vertex ClipTest
//
//  On input:
//  if (CLIP.PreMapped)
//    x,y are viewport mapped
//    z is NDC ([0,1] is visible)
//  else
//    x,y,z are NDC (post-perspective divide)
//  w is always 1/w

//
// Initialize outCodes to “inside”
//
outCode[*] = 0

//
// Check if w is NaN
// Any object containing one of these “bad” vertices will likely be discarded
 //
 #ifdef (DevBW-E0 || DevCL-B)
 if (ISNAN(w)|| UserClipFlag[7])
 #elseif (DevCTG)
 if (ISNAN(w))
 #elseif (DevIL+)
 if (ISNAN(homogeneous x,y,z,w or enabled ClipDistance value)
 #endif
{
   outCode[BAD] = 1
}
```

```
//
//  If 1/w is negative, w is negative and therefore outside of the w=0 plane
//
//
rhw_neg = ISNEG(rhw)
if (rhw_neg)
{
#ifdef (PreDevBW-E0 || DevCL-A)
   outCode[VP_XMIN] = 1
   outCode[VP_XMAX] = 1
   outCode[VP_YMIN] = 1
   outCode[VP_YMAX] = 1
   outCode[VP_ZMIN] = 1
   outCode[VP_ZMAX] = 1
   outCode[GB_XMIN] = 1
   outCode[GB_XMAX] = 1
   outCode[GB_YMIN] = 1
   outCode[GB_YMAX] = 1
   goto UserClipFlags
 #elseifdef (DevCTG+)
   outCode[NEGW] = 1
 #endif
}

//
// View Volume Clip Test
//   If Premapped, the 2D viewport is defined in screen space
//     otherwise the canonical NDC viewvolume applies ([-1,1])
//
if (CLIP_STATE.PreMapped)
```

```
{
   vp_XMIN = CLIP_STATE.VP_XMIN
   vp_XMAX = CLIP_STATE.VP_XMAX
   vp_YMIN = CLIP_STATE.VP_YMIN
   vp_YMAX = CLIP_STATE.VP_YMAX
} else {
   vp_XMIN = -1.0f
   vp_XMAX = +1.0f
   vp_YMIN = -1.0f
   vp_YMAX = +1.0f
}
 if (CLIP_STATE.ViewportXYClipTestEnable) {
   outCode[VP_XMIN] = (x < vp_XMIN)
   outCode[VP_XMAX] = (x > vp_XMAX)
   outCode[VP_YMIN] = (y < vp_YMIN)
   outCode[VP_YMAX] = (y > vp_YMAX)
  #ifdef (DevBW-E0)
    if (rhw_neg) {
       outCode[VP_XMIN] = (x >= vp_XMIN)
       outCode[VP_XMAX] = (x <= vp_XMAX)
       outCode[VP_YMIN] = (y >= vp_XMIN)
       outCode[VP_YMAX] = (y <= vp_XMAX)
    }
 #endif
  #ifdef (DevCTG+)
    if (rhw_neg) {
       outCode[VP_XMIN] = (x > vp_XMIN)
       outCode[VP_XMAX] = (x < vp_XMAX)
       outCode[VP_YMIN] = (y > vp_XMIN)
       outCode[VP_YMAX] = (y < vp_XMAX)
    }
```

```
  #endif
  }
  if (CLIP_STATE.ViewportZClipTestEnable) {
    if (CLIP_STATE.APIMode == APIMODE_D3D) {
        vp_ZMIN = 0.0f
        vp_ZMAX = 1.0f
    } else {  // OGL
        vp_ZMIN = -1.0f
        vp_ZMAX = 1.0f
    }
    outCode[VP_ZMIN] = (z < vp_ZMIN)
    outCode[VP_ZMAX] = (z > vp_ZMAX)
   #ifdef (DevBW-E0)
     if (rhw_neg) {
        outCode[VP_ZMIN] = (z >= vp_ZMIN)
        outCode[VP_ZMAX] = (z <= vp_ZMAX)
     }
  #endif
   #ifdef (DevCTG+)
     if (rhw_neg) {
        outCode[VP_ZMIN] = (z > vp_ZMIN)
        outCode[VP_ZMAX] = (z < vp_ZMAX)
     }
  #endif
 }

 //
 // Guardband Clip Test
 //
 if {CLIP_STATE.GuardbandClipTestEnable) {
     gb_XMIN = CLIP_STATE.Viewport[vpindex].GB_XMIN
```

```
    gb_XMAX = CLIP_STATE.Viewport[vpindex].GB_XMAX
    gb_YMIN = CLIP_STATE.Viewport[vpindex].GB_YMIN
    gb_YMAX = CLIP_STATE.Viewport[vpindex].GB_YMAX
    outCode[GB_XMIN] = (x < gb_XMIN)
    outCode[GB_XMAX] = (x > gb_XMAX)
    outCode[GB_YMIN] = (y < gb_YMIN)
    outCode[GB_YMAX] = (y > gb_YMAX)
  #ifdef (DevBW-E0)
    if (rhw_neg) {
        outCode[GB_XMIN] = (x >= gb_XMIN)
        outCode[GB_XMAX] = (x <= gb_XMAX)
        outCode[GB_YMIN] = (y >= gb_YMIN)
        outCode[GB_YMAX] = (y <= gb_YMAX)
    }
 #endif
  #ifdef (DevCTG+)
    if (rhw_neg) {
        outCode[GB_XMIN] = (x > gb_XMIN)
        outCode[GB_XMAX] = (x < gb_XMAX)
        outCode[GB_YMIN] = (y > gb_YMIN)
        outCode[GB_YMAX] = (y < gb_YMAX)
    }
 #endif
}

//
// Handle case where either VP or GB disabled (but not both)
// In this case, the disabled set take on the outcodes of the enabled set
//
if (CLIP_STATE.ViewportXYClipTestEnable && !CLIP_STATE.GuardbandClipTestEnable) {
    outCode[GB_XMIN] = outCode[VP_XMIN]
```

```
        outCode[GB_XMAX] = outCode[VP_XMAX]

        outCode[GB_YMIN] = outCode[VP_YMIN]

        outCode[GB_YMAX] = outCode[VP_YMAX]

    } else if (!CLIP_STATE.ViewportXYClipTestEnable &&
CLIP_STATE.GuardbandClipTestEnable) {

        outCode[VP_XMIN] = outCode[GB_XMIN]

        outCode[VP_XMAX] = outCode[GB_XMAX]

        outCode[VP_YMIN] = outCode[GB_YMIN]

        outCode[VP_YMAX] = outCode[GB_YMAX]

    }


    //

    // X/Y/Z NaN Handling

    //

    xyorgben = (CLIP_STATE.ViewportXYClipTestEnable ||
CLIP_STATE.GuardbandClipTestEnable)

    if (isNAN(x)) {

          outCode[GB_XMIN] = xyorgben

          outCode[GB_XMAX] = xyorgben

         outCode[VP_XMIN] = xyorgben

         outCode[VP_XMAX] = xyorgben

     }


    if (isNAN(y)) {

          outCode[GB_YMIN] = xyorgben

          outCode[GB_YMAX] = xyorgben

         outCode[VP_YMIN] = xyorgben

         outCode[VP_YMAX] = xyorgben

    }


    if (isNaN) {

          outCode[VP_ZMIN] = CLIP_STATE.ViewportZClipTestEnable
```

```
        outCode[VP_ZMAX] = CLIP_STATE.ViewportZClipTestEnable

   }



   //

   // UserClipFlags

   //

   ExamineUCFs

   for (i=0; i<7; i++)

   {

      outCode[UC0+i] = userClipFlag[i] &
CLIP_STATE.UserClipFlagsClipTestEnableBitmask[i]

    }

    #ifdef (DevBW-E0 || DevCL-B)

      outCode[UC7] = rhw_neg & CLIP_STATE.UserClipFlagsClipTestEnableBitmask[7]

    #else

      outCode[UC7] = userClipFlag[i] &
CLIP_STATE.UserClipFlagsClipTestEnableBitmask[7]

    #endif
```

5.5 Object Staging

The CLIP unit's Object Staging Buffer (OSB) accepts streams of input vertex information packets, along with each vertex's VertexClipTest result (outCode). This information is buffered until a complete object or the last vertex of the primitive topology is received. The OSB then performs the ClipDetermination function on the object vertices, and takes the actions required by the results of that function.

5.5.1 Partial Object Removal

The OSB is responsible for removing incomplete LINESTRIP and TRISTRIP objects that it may receive from the preceding stage (GS). Partial object removal is not supported for other primitive types due to either (a) the GS is not permitted to output those primitive types (e.g., primitives with adjacency info), and the VF unit will have removed the partial objects as part of 3DPRIMITIVE processing, or (b) although the GS thread is allowed to output the primitive type (e.g., LINELIST), it is assumed that the GS kernel will be correctly implemented to avoid outputting partial objects (or pipeline behavior is UNDEFINED). In short, CLIP unit partial object removal is only provided for the cases where the GS shader programmer is able to generate partial objects.

An object is considered ‘partial’ if the last vertex of the primitive topology is encountered (i.e., PrimEnd is set) before a complete set of vertices for that object have been received. Given that only LINESTRIP and TRISTRIP primitive types are subject to CLIP unit partial object removal, the only supported cases of partial objects are 1-vertex LINESTRIPs and 1 or 2-vertex TRISTRIPs.

[errata DevSNB]: Possible hang if final output from GS kernel is 2 vertex triangle. If it is possible for the final output from GS kernel to be a 2 vertex triangle, then have the GS kernel always output an extra single vertex triangle as the final output.

5.5.2 ClipDetermination Function

In ClipDetermination, the vertex outcodes of the primitive are combined in order to determine the clip status of the object (TR: trivially reject; TA: trivial accept; MC: must clip; BAD: invalid coordinate). Only those vertices included in the object are examined (3 vertices for a triangle, 2 for a line, and 1 for a point). The outcode bit arrays for the vertices are separately ANDed (intersection) and ORed (union) together (across vertices, not within the array) to yield objANDCode and objORCode bit arrays.

TR/TA against interesting boundary subsets are then computed. The TR status is computed as the logical OR of the appropriate objANDCode bits, as the vertices need only be outside of one common boundary to be trivially rejected. The TA status is computed as the logical NOR of the appropriate objORCode bits, as any vertex being outside of any of the boundaries prevents the object from being trivially accepted.

If any vertex contains a BAD coordinate, the object is considered BAD and any computed TR/TA results will effectively be ignored in the final action determination.

Next, the boundary subset TR/TA results are combined to determine an overall status of the object. If the object is TR against any viewport or enabled UC plane, the object is considered TR. Note that, by definition, being TR against a VPXY boundary implies that the vertices will be TR agains the corresponding GB boundary, so computing `TR_GB` is unnecessary.

The treatment of the UCF outcodes is conditional on the UserClipFlags MustClip Enable state. If DISABLED, an object that is not TR against the UCFs is considered TA against them. Put another way, objects will only be culled (not clipped) with respect to the UCFs. If ENABLED, the UCF outcodes are treated like the other outcodes, in that they are used to determine TR, TA or MC status, and an object can be passed to a CLIP thread simply based on it straddling a UCF.

Finally, the object is considered MC if it is neither TR or TA.

The following logic is used to compute the final TR, TA, and MC status.

```
//
// ClipDetermination
//


//
// Compute objANDCode and objORCode
//
```

```
   switch (object type) {

   case POINT:

   {

      objANDCode[…] = v0.outCode[…]

      objORCode[…]  = v0.outCode[…]

   } break

   case LINE:

   {

      objANDCode[…] = v0.outCode[…] & v1.outCode[…]

      objORCode[…]  = v0.outCode[…] | v1.outCode[…]

   } break

   case TRIANGLE:

   {

      objANDCode[…] = v0.outCode[…] & v1.outCode[…] & v2.outCode[…]

      objORCode[…]  = v0.outCode[…] | v1.outCode[…] | v2.outCode[…]

   } break


   //

   // Determine TR/TA against interesting boundary subsets

   //

   TR_VPXY = (objANDCode[VP_L] | objANDCode[VP_R] | objANDCode[VP_T] |
objANDCode[VP_B])

   TR_GB   = (objANDCode[GB_L] | objANDCode[GB_R] | objANDCode[GB_T] |
objANDCode[GB_B])

   TA_GB   = !(objORCode[GB_L] | objORCode[GB_R] | objORCode[GB_T] | objORCode[GB_B])

   TA_VPZ  = !(objORCode[VP_N] | objORCode[VP_Z])

   TR_VPZ  = (objANDCode[VP_N] | objANDCode[VP_Z])

   TA_UC   = !(objORCode[UC0] | objORCode[UC1] | … | objORCode[UC7])

   TR_UC   = (objANDCode[UC0] | objANDCode[UC1] | … | objANDCode[UC7])

   BAD     = objORCode[BAD]

   #ifdef (DevCTG+)

   TA_NEGW  = !objORCode[NEGW]
```

```
TR_NEGW  = objANDCode[NEGW]
#endif

//
// Trivial Reject
//
//    An object is considered TR if all vertices are TR against any common boundary
//    Note that this allows the case of the VPXY being outside the GB
//
#ifdef (DevCTG+)
TR = TR_GB || TR_VPXY || TR_VPZ || TR_UC || TR_NEGW
#else
TR = TR_GB || TR_VPXY || TR_VPZ || TR_UC
#endif

//
// Trivial Accept
//
// For an object to be TA, it must be TA against the VPZ and GB, not TR,
//   and considered TA against the UC planes and (DevCTG+) NEGW
// If the UCMC mode is disabled, an object is considered TA against the UC
//   as long as it isn't TR against the UC.
// If the UCMC mode is enabled, then the object really has to be TA against the UC
//   to be considered TA
// In this way, enabling the UCMC mode will force clipping if the object is neither
//   TA or TR against the UC
//
#ifdef (DevCTG+)
TA = !TR && TA_GB && TA_VPZ && TA_NEGW
#else
TA = !TR && TA_GB && TA_VPZ
```

```
#endif

UCMC = CLIP_STATE.UserClipFlagsMustClipEnable

TA = TA && ( (UCMC && TA_UC) || (!UCMC && !TR_UC) )


//

// MustClip

// This is simply defined as not TA or TR

// Note that exactly one of TA, TR and MC will be set

//

MC = !(TA || TR)
```

5.5.3 ClipMode

The ClipMode state determines what action the CLIP unit takes given the results of ClipDetermination. The possible actions are:

- PASSTHRU: Pass the object directly down the pipeline. A CLIP thread is not spawned.
- DISCARD: Remove the object from the pipeline and dereference object vertices as required (i.e., dereferencing will not occur if the vertices are shared with other objects).
- SPAWN: Pass the object to a CLIP thread. In the process of initiating the thread, the object vertices may be dereferenced.

The following logic is used to determine what to do with the object (PASSTHRU or DISCARD or SPAWN).

```
//

// Use the ClipMode to determine the action to take

//

switch (CLIP_STATE.ClipMode) {

  case NORMAL: {

    PASSTHRU = TA && !BAD

    DISCARD  = TR || BAD

    SPAWN    = MC && !BAD

  }
```

```
    case CLIP_ALL: {
     PASSTHRU = 0
     DISCARD  = 0
     SPAWN    = 1
    }
    case CLIP_NOT_REJECT: {
     PASSTHRU = 0
     DISCARD  = TR || BAD
     SPAWN    = !(TR || BAD)
    }
    case REJECT_ALL: {
     PASSTHRU = 0
     DISCARD  = 1
     SPAWN    = 0
    }
    case ACCEPT_ALL: {
     PASSTHRU = !BAD
     DISCARD  = BAD
     SPAWN    = 0
    }
  } endswitch

  #ifdef (DevBW-E0 || DevCL-B)
  if (BAD && CLIP_STATE.ClipMode != REJECT_ALL) {
      DISCARD = 0
      SPAWN = 1
  }
  #endif
```

5.5.3.1 NORMAL ClipMode

In NORMAL mode, objects will be discarded if TR or BAD, passed through if TA, and passed to a CLIP thread if MC. Those mode is typically used when the CLIP kernel is only used to perform 3D Clipping (the expected usage model).

5.5.3.2 CLIP_ALL ClipMode

In CLIP_ALL mode, all objects (regardless of classification) will be passed to CLIP threads. Note that this includes BAD objects. This mode can be used to perform arbitrary processing in the CLIP thread, or as a backup if for some reason the CLIP unit fixed functions (VertexClipTest, ClipDetermination) are not sufficient for controlling 3D Clipping.

5.5.3.3 CLIP_NON_REJECT ClipMode

This mode is similar to CLIP_ALL mode, but TR and BAD objects are discarded an all other (TA, MC) objects are passed to CLIP threads. Usage of this mode assumes that the CLIP unit fixed functions (VertexClipTest, ClipDetermination) are sufficient at least in respect to determining trivial reject.

5.5.3.4 REJECT_ALL ClipMode

In REJECT_ALL mode, all objects (regardless of classification) are discarded. This mode effectively clips out all objects.

5.5.3.5 ACCEPT_ALL ClipMode

In ACCEPT_ALL mode, all non-BAD objects are passed directly down the pipeline. This mode partially disables the CLIP stage. BAD objects will still be discarded, and incomplete primitives (generated by a GS thread) will be discarded.

Primitive topologies with adjacency are also handled, in that the adjacent-only vertices are dereferenced and only non-adjacent objects are passed down the pipeline. This condition can arise when primitive topologies with adjacency are generated but the GS stage is disabled. If this condition is allowed, the CLIP stage must not be completely disabled – as this would allow adjacent vertices to pass through the CLIP stage and lead to UNPREDICATBLE results as the rest of the pipeline does not comprehend adjacency.

5.6 Object Pass-Through

Depending on ClipMode, objects may be passed directly down the pipeline. The PrimTopologyType associated with the output objects may differ from the input PrimTopologyType, as shown in the table below.

Programming Note: The CLIP unit does <u>not</u> tolerate primitives with adjacency that have “dangling vertices”. This should not be an issue under normal conditions, as the VF unit will not generate these sorts of primitives and the GS thread is restricted (though by specification only) to not output these sorts of primitives.

Input PrimTopologyType	Pass-Through Output PrimTopologyType	Notes
POINTLIST	POINTLIST	
POINTLIST_BF	POINTLIST_BF	
LINELIST	LINELIST	
LINELIST_ADJ	LINELIST	Adjacent vertices removed.
LINESTRIP	LINESTRIP	
LINESTRIP_ADJ	LINESTRIP	Adjacent vertices removed.
LINESTRIP_BF	LINESTRIP_BF	
LINESTRIP_CONT	LINESTRIP_CONT	
LINESTRIP_CONT_BF	LINESTRIP_CONT_BF	
LINELOOP	N/A	Not supported after GS.
TRILIST	TRILIST	
RECTLIST	RECTLIST	
TRILIST_ADJ	TRILIST	Adjacent vertices removed.
TRISTRIP	TRISTRIP or TRISTRIP_REV	Depends on where the incoming strip is broken (if at all) by discarded or clipped objects See Tristrip Clipping Errata subsection.
TRISTRIP_REV	TRISTRIP or TRISTRIP_REV	Depends on where the incoming strip is broken (if at all) by discarded or clipped objects See Tristrip Clipping Errata subsection.
TRISTRIP_ADJ	TRISTRIP or TRISTRIP_REV	Depends on where the incoming strip is broken (if at all) by discarded or clipped objects Adjacent vertices removed. See Tristrip Clipping Errata subsection.
TRIFAN	TRIFAN	
TRIFAN_NOSTIPPLE	TRIFAN_NOSTIPPLE	
POLYGON	POLYGON	

Input PrimTopologyType	Pass-Through Output PrimTopologyType	Notes
QUADLIST	N/A	Not supported after GS.
QUADSTRIP	N/A	Not supported after GS.

5.7 Primitive Output

(This section refers to output from the CLIP unit to the pipeline, not output from the CLIP thread)

The CLIP unit will output primitives (either passed-through or generated by a CLIP thread) in the proper order. This includes the buffering of a concurrent CLIP thread's output until the preceding CLIP thread terminates. Note that the requirement to buffer subsequent CLIP thread output until the preceding CLIP thread terminates has ramifications on determining the number of VUEs allocated to the CLIP unit and the number of concurrent CLIP threads allowed.

5.8 Other Functionality

5.8.1 Statistics Gathering

The CLIP unit includes logic to assist in the gathering of certain pipeline statistics, primarily in support of the Asynchronous Query function of the D3D APIs. The statistics take the form of MI counter registers (see *Memory Interface Registers*), where the CLIP unit provides signals causing those counters to increment.

Software is responsible for controlling (enabling) these counters in order to provide the required statistics at the DDI level. For example, software might need to disable the statistics gathering before submitting non-API-visible objects (e.g., RECTLISTs) for processing.

The CLIP unit must be ENABLED (via the **CLIP Enable** bit of PIPELINED_STATE_POINTERS) in order to it to affect the statistics counters. This might lead to a pathological case where the CLIP unit needs to be ENABLED simply to provide statistics gathering. If no clipping functionality is desired, **Clip Mode** can be set to ACCEPT_ALL to effectively inhibit clipping while leaving the CLIP stage ENABLED.

The two statistics the CLIP unit affects (if enabled) are:

- CL_INVOCATION_COUNT:
    - Incremented for every object received from the GS stage.

5.8.1.1 CL_INVOCATION_COUNT

If the **Statistics Enable** bit (CLIP_STATE) is set, the CLIP unit increments the CL_INVOCATION_COUNT register for every complete object received from the GS stage.

In order to maintain a count of application-generated objects, software will need to clear the CLIP unit's **Statistic Enable** whenever driver-generated objects are rendered.

6. Strips and Fans (SF) Stage

6.1 Overview

The Strips and Fan (SF) stage of the 3D pipeline is responsible for performing "setup" operations required to rasterize 3D objects.

[DevSNB+]: This functionality is handled completely in hardware, and the SF unit no longer has the ability to spawn threads.

6.1.1 Inputs from CLIP

The following table describes the per-vertex inputs passed to the SF unit from the previous (CLIP) stage of the pipeline.

Table 14. SF's Vertex Pipeline Inputs

Variable	Type	Description
primType	enum	Type of primitive topology the vertex belongs to. See Table 15 for a list of primitive types supported by the SF unit. See *3D Pipeline* for descriptions of these topologies. Notes: The CLIP unit will convert any primitive with adjacency (3DPRIMxxx_ADJ) it receives from the pipeline into the corresponding primitive without adjacency (3DPRIMxxx). QUADLIST, QUADSTRIP, LINELOOP primitives are not supported by the SF unit. Software must use a GS thread to convert these to some other (supported) primitive type. [DevSNB+] If an object is clipped by the hardware clipper, the CLunit would force this field to "3DPRIM_POLYGON". SFunit would process this incoming object just as it would any other "3DPRIM_POLYGON". SFunit selects vertex 0 as the provoking vertex.
primStart,primEnd	boolean	Indicate vertex's position within the primitive topology
vInX[]	float	Vertex X position (screen space or NDC space)
vInY[]	float	Vertex Y position (screen space or NDC space)
vInZ[]	float	Vertex Z position (screen space or NDC space)

Variable	Type	Description
vInInvW[]	float	Reciprocal of Vertex homogeneous (clip space) W
hVUE[]	URB address	Points to the vertex's data stored in the URB (one VUE handle per vertex)
renderTargetArrayIndex	uint	Index of the render target (array element or 3D slice), clamped to 0 by the GS unit if the max value was exceeded. If this vertex is the leading vertex of an object within the primitive topology, this value will be associated with that object in subsequent processing.
viewPortIndex	uint	Index of a viewport transform matrix within the SF_VIEWPORT structure used to perform Viewport Transformation on object vertices and scissor operations on an object. If this vertex is the leading vertex of an object within the primitive topology, this value will be associated with that object in the Viewport Transform and Scissor subfunctions, otherwise the value is ignored. Note that for primitive topologies with vertices shared between objects, this means a shared vertex may be subject to multiple Viewport Transformation operations if the viewPortIndex varies within the topology.
pointSize	uint	If this vertex is within a POINTLIST[_BF] primitive topology, this value specifies the screen space size (width,height) of the square point to be rasterized about the vertex position. Otherwise the value is ignored.

6.1.2 Attribute Setup/Interpolation Process

6.1.2.1 Attribute Setup/Interpolation Process [DevSNB+]

Computation of all parameters needed is performed by hardware as there is no setup thread.

6.1.3 Outputs to WM

The outputs from the SF stage to the WM stage are mostly comprised of implementation-specific information required for the rasterization of objects. The types of information is summarized below, but as the interface is not exposed to software a detailed discussion is not relevant to this specification.

- PrimType of the object
- VPIndex, RTAIndex associated with the object
- **[DevSNB+]:** Coefficients for Z, 1/W, perspective and non-perspective b1 and b2 per vertex, and attribute vertex deltas a0, a1, and a2 per attribute.
- Information regarding the X,Y extent of the object (e.g., bounding box, etc.)
- Edge or line interpolation information (e.g., edge equation coefficients, etc.)
- Information on where the WM is to start rasterization of the object

- Object orientation (front/back-facing)
- Last Pixel indication (for line drawing)

6.2 Primitive Assembly

The first subfunction within the SF unit is *Primitive Assembly*. Here 3D primitive vertex information is buffered and, when a sufficient number of vertices are received, converted into basic 3D objects which are then passed to the Viewport Transformation subfunction.

The number of vertices passed with each primitive is constrained by the primitive type and must conform to Table 15. Passing any other number of vertices results in UNDEFINED behavior. Note that this restriction only applies to primitive output by GS threads (which is under control of the GS kernel). See the Vertex Fetch chapter for details on how the VF unit automatically removes incomplete objects resulting from processing a 3DPRIMITIVE command.

Table 15. SF-Supported Primitive Types & Vertex Count Restrictions

primType	VertexCount Restriction
3DPRIM_TRILIST	nonzero multiple of 3
3DPRIM_TRISTRIP 3DPRIM_TRISTRIP_REVERSE	>=3
3DPRIM_TRIFAN 3DPRIM_TRIFAN_NOSTIPPLE 3DPRIM_POLYGON	>=3
3DPRIM_LINELIST	nonzero multiple of 2
3DPRIM_LINESTRIP 3DPRIM_LINESTRIP_CONT 3DPRIM_LINESTRIP_BF 3DPRIM_LINESTRIP_CONT_BF	>=2
3DPRIM_RECTLIST	nonzero multiple of 3
3DPRIM_POINTLIST 3DPRIM_POINTLIST_BF	nonzero

The 3D object types are listed in the table below.

Table 16. 3D Object Types

objectType	generated by *primType*	Vertices/Object
3DOBJ_POINT	3DPRIM_POINTLIST 3DPRIM_POINTLIST_BF	1
3DOBJ_LINE	3DPRIM_LINELIST 3DPRIM_LINESTRIP 3DPRIM_LINESTRIP_CONT 3DPRIM_LINESTRIP_BF 3DPRIM_LINESTRIP_CONT_BF	2
3DOBJ_TRIANGLE	3DPRIM_TRILIST 3DPRIM_TRISTRIP 3DPRIM_TRISTRIP_REVERSE 3DPRIM_TRIFAN 3DPRIM_TRIFAN_NOSTIPPLE 3DPRIM_POLYGON	3
3DOBJ_RECTANGLE	3DPRIM_RECTLIST	3 (expanded to 4 in RectangleCompletion)

The outputs of Primitive Decomposition are listed in the following table.

Table 17. Primitive Decomposition Outputs

Variable	Type	Description
objectType	enum	Type of object. See Table 16
nV	uint	The number of object vertices passed to Object Setup. See Table 16
v[0..nV-1]*	various	Data arrays associated with object vertices. Data in the array consists of X, Y, Z, invW and a pointer to the other vertex attributes. These additional attributes are not used by directly by the 3D fixed functions but are made available to the SF thread. The number of valid vertices depends on the object type. See Table 16
invertOrientation	enum	Indicates whether the orientation (CW or CCW winding order) of the vertices of a triangle object should be inverted. Ignored for non-triangle objects.
backFacing	enum	Valid only for points and line objects, indicates a back facing object. This is used later for culling.

Variable	Type	Description
provokingVtx	uint	Specifies the index (into the *v[]* arrays) of the vertex considered the "provoking" vertex (for flat shading). The selection of the provoking vertex is programmable via SF_STATE (**xxx Provoking Vertex Select** state variables.)
polyStippleEnable	boolean	TRUE if Polygon Stippling is enabled. FALSE for TRIFAN_NOSTIPPLE. Ignored for non-triangle objects.
continueStipple	boolean	Only applies to line objects. TRUE if Line Stippling should be continued (i.e., not reset) from where the previous line left off. If FALSE, Line Stippling is reset for each line object.
renderTargetIndex	uint	Index of the render target (array element or 3D slice), clamped to 0 by the GS unit if the max value was exceeded. This value is simply passed in SF thread payloads and not used within the SF unit.
viewPortIndex	uint	Index of a viewport transform matrix within the SF_VIEWPORT structure used to perform Viewport Transformation on object vertices and scissor operations on an object.
pointSize	unit	For point objects, this value specifies the screen space size (width,height) of the square point to be rasterized about the vertex position. Otherwise the value is ignored.

The following table defines, for each primitive topology type, which vertex's VPIndex/RTAIndex applies to the objects within the topology.

Table 18. VPIndex/RTAIndex Selection

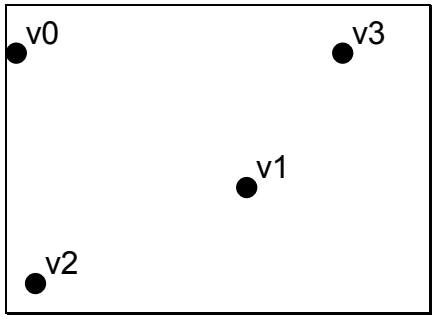
PrimTopologyType	Viewport Index Usage
POINTLIST POINTLIST_BF	Each vertex supplies the VPIndex for the corresponding point object
LINELIST	The leading vertex of each line supplies the VPIndex for the corresponding line object. V0.VPIndex → Line(V0,V1) V2.VPIndex → Line(V2,V3) …
LINESTRIP LINESTRIP_BF LINESTRIP_CONT LINESTRIP_CONT_BF	The leading vertex of each line segment supplies the VPIndex for the corresponding line object. V0.VPIndex → Line(V0,V1) V1.VPIndex → Line(V1,V2) … NOTE: If the VPIndex changes within the topology, shared vertices will be processed (mapped) multiple times.
TRILIST	The leading vertex of each triangle/rect supplies the VPIndex for the

PrimTopologyType	Viewport Index Usage
RECTLIST	corresponding triangle/rect objects. V0.VPIndex → Tri(V0,V1,V2) V3.VPIndex → Tri(V3,V4,V5) …
TRISTRIP TRISTRIP_REVERSE	The leading vertex of each triangle supplies the VPIndex for the corresponding triangle object. V0.VPIndex → Tri(V0,V1,V2) V1.VPIndex → Tri(V1,V2,V3) … NOTE: If the VPIndex changes within the primitive, shared vertices will be processed (mapped) multiple times.
TRIFAN TRIFAN_NOSTIPPLE POLYGON	The first vertex (V0) supplies the VPIndex for all triangle objects.

6.2.1 Point List Decomposition

The 3DPRIM_POINTLIST and 3DPRIM_POINTLIST_BACKFACING primitives specify a list of independent points.

Figure 6-1. 3DPRIM_POINTLIST Primitive



The decomposition process divides the list into a series of basic 3DOBJ_POINT objects that are then passed individually and in order to the Object Setup subfunction. The *provokingVertex* of each object is, by definition, v[0].

Points have no winding order, so the primitive command is used to explicitly state whether they are back-facing or front-facing points. Primitives of type 3DPRIM_POINTLIST_BACKFACING are decomposed exactly the same way as 3DPRIM_POINTLIST primitives, but the *backFacing* variable is set for resulting point objects being passed on to object setup.

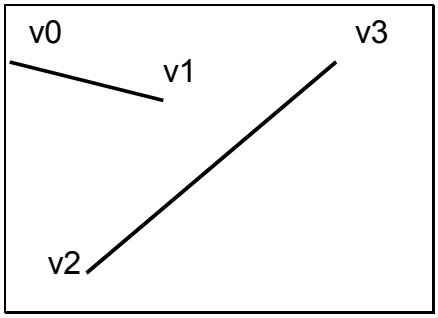
```
PointListDecomposition() {

        objectType = 3DOBJ_POINT
```

```
        nV = 1
        provokingVtx = 0
        if (primType == 3DPRIM_POINTLIST)
                backFacing = FALSE
        else  // primType == 3DPRIM_POINTLIST_BACKFACING
                backFacing = TRUE
        for each (vertex I in [0..vertexCount-1]) {
                v[0] ← vIn[i]             // copy all arrays (e.g., v[]X, v[]Y, etc.)
                ObjectSetup()
        }
}
```

6.2.2 Line List Decomposition

The 3DPRIM_LINELIST primitive specifies a list of independent lines.

Figure 6-2. 3DPRIM_LINELIST Primitive



The decomposition process divides the list into a series of basic 3DOBJ_LINE objects that are then passed individually and in order to the Object Setup stage. The lines are generated with the following object vertex order: v0, v1; v2, v3; and so on. The *provokingVertex* of each object is taken from the **Line List/Strip Provoking Vertex Select** state variable, as programmed via SF_STATE.

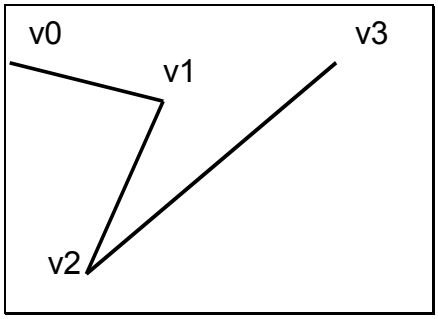
```
LineListDecomposition() {
        objectType = 3DOBJ_LINE
        nV = 2
        provokingVtx = Line List/Strip Provoking Vertex Select
```

```
        continueStipple = FALSE
        for each (vertex I in [0..vertexCount-2] by 2) {
                v[0] arrays ← vIn[i] arrays
                v[1] arrays ← vIn[i+1] arrays
                ObjectSetup()
        }
}
```

6.2.3 Line Strip Decomposition

The 3DPRIM_LINESTRIP, 3DPRIM_LINESTRIP_CONT, 3DPRIM_LINESTRIP_BF, and 3DPRIM_LINESTRIP_CONT_BF primitives specify a list of connected lines.

Figure 6-3. 3DPRIM_LINESTRIP_xxx Primitive



The decomposition process divides the strip into a series of basic 3DOBJ_LINE objects that are then passed individually and in order to the Object Setup stage. The lines are generated with the following object vertex order: v0,v1; v1,v2; and so on. The *provokingVertex* of each object is taken from the **Line List/Strip Provoking Vertex Select** state variable, as programmed via SF_STATE.

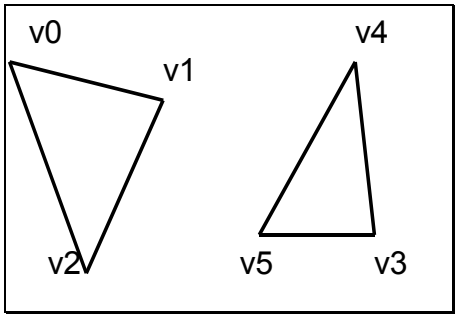
Lines have no winding order, so the primitive command is used to explicitly state whether they are back-facing or front-facing lines. Primitives of type 3DPRIM_LINESTRIP[_CONT]_BF are decomposed exactly the same way as 3DPRIM_LINESTRIP[_CONT] primitives, but the *backFacing* variable is set for the resulting line objects being passed on to object setup. Likewise 3DPRIM_LINESTRIP_CONT[_BF] primitives are decomposed identically to basic line strips, but the *continueStipple* variable is set to true so that the line stipple pattern will pick up from where it left off with the last line primitive, rather than being reset.

```
LineStripDecomposition() {
        objectType = 3DOBJ_LINE
        nV = 2
        provokingVtx = Line List/Strip Provoking Vertex Select
        if (primType == 3DPRIM_LINESTRIP) {
                backFacing = FALSE
                continueStipple = FALSE
        } else if (primType == 3DPRIM_LINESTRIP_BF) {
                backFacing = TRUE
                continueStipple = FALSE
        } else if (primType == 3DPRIM_LINESTRIP_CONT) {
                backFacing = FALSE
                continueStipple = TRUE
        } else if (primType == 3DPRIM_LINESTRIP_CONT_BF) {
                backFacing = TRUE
                continueStipple = TRUE
        }
        for each (vertex I in [0..vertexCount-1]) {
                v[0] arrays ← vIn[i] arrays
                v[1] arrays ← vIn[i+1] arrays
                ObjectSetup()
                continueStipple = TRUE
        }
}
```

6.2.4 Triangle List Decomposition

The 3DPRIM_TRILIST primitive specifies a list of independent triangles.

Figure 6-4. 3DPRIM_TRILIST Primitive



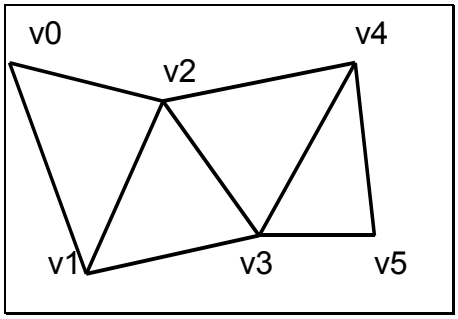
The decomposition process divides the list into a series of basic 3DOBJ_TRIANGLE objects that are then passed individually and in order to the Object Setup stage. The triangles are generated with the following object vertex order: v0,v1,v2; v3,v4,v5; and so on. The *provokingVertex* of each object is taken from the **Triangle List/Strip Provoking Vertex Select** state variable, as programmed via SF_STATE.

```
TriangleListDecomposition() {
        objectType = 3DOBJ_TRIANGLE
        nV = 3
        invertOrientation = FALSE
        provokingVtx = Triangle List/Strip Provoking Vertex Select
        polyStippleEnable = TRUE
        for each (vertex I in [0..vertexCount-3] by 3) {
                v[0] arrays ← vIn[i] arrays
                v[1] arrays ← vIn[i+1] arrays
                v[2] arrays ← vIn[i+2] arrays
                ObjectSetup()
        }
}
```

6.2.5 Triangle Strip Decomposition

The 3DPRIM_TRISTRIP and 3DPRIM_TRISTRIP_REVERSE primitives specify a series of triangles arranged in a strip, as illustrated below.

Figure 6-5. 3DPRIM_TRISTRIP[_REVERSE] Primitive



The decomposition process divides the strip into a series of basic 3DOBJ_TRIANGLE objects that are then passed individually and in order to the Object Setup stage. The triangles are generated with the following object vertex order: v0,v1,v2; v1,v2,v3; v2,v3,v4; and so on. Note that the *winding order* of the vertices alternates between CW (clockwise), CCW (counter-clockwise), CW, etc. The *provokingVertex* of each object is taken from the **Triangle List/Strip Provoking Vertex Select** state variable, as programmed via SF_STATE.

The 3D pipeline uses the winding order of the vertices to distinguish between front-facing and back-facing triangles (see Triangle Orientation (Face) Culling below). Therefore, the 3D pipeline must account for the alternation of winding order in strip triangles. The *invertOrientation* variable is generated and used for this purpose.

To accommodate the situation where the driver is forced to break an input strip primitive into multiple tristrip primitive commands (e.g., due to ring or batch buffer size restrictions), two tristrip primitive types are supported. 3DPRIM_TRISTRIP is used for the initial section of a strip, and wherever a continuation of a strip starts with a triangle with a CW winding order. 3DPRIM_TRISTRIP_REVERSE is used for a continuation of a strip that starts with a triangle with a CCW winding order.

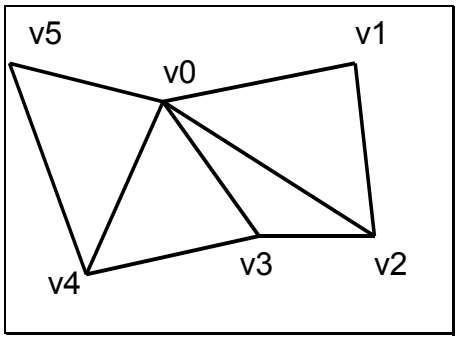
```
TriangleStripDecomposition() {

        objectType = 3DOBJ_TRIANGLE

        nV = 3

        provokingVtx = Triangle List/Strip Provoking Vertex Select

        if (primType == 3DPRIM_TRISTRIP)

                invertOrientation = FALSE

        else  // primType == 3DPRIM_TRISTRIP_REVERSE

                invertOrientation = TRUE

        polyStippleEnable = TRUE
```

```
        for each (vertex I in [0..vertexCount-3]) {
                v[0] arrays ← vIn[i] arrays
                v[1] arrays ← vIn[i+1] arrays
                v[2] arrays ← vIn[i+2] arrays
                ObjectSetup()
                invertOrientation = ! invertOrientation
        }
}
```

6.2.6 Triangle Fan Decomposition

The 3DPRIM_TRIFAN and 3DPRIM_TRIFAN_NOSTIPPLE primitives specify a series of triangles arranged in a fan, as illustrated below.

Figure 6-6. 3DPRIM_TRIFAN Primitive



The decomposition process divides the fan into a series of basic 3DOBJ_TRIANGLE objects that are then passed individually and in order to the Object Setup stage. The triangles are generated with the following object vertex order: v0,v1,v2; v0,v2,v3; v0,v3,v4; and so on. As there is no alternation in the vertex winding order, the *invertOrientation* variable is output as FALSE unconditionally. The *provokingVertex* of each object is taken from the **Triangle Fan Provoking Vertex** state variable, as programmed via SF_STATE.

Primitives of type 3DPRIM_TRIFAN_NOSTIPPLE are decomposed exactly the same way, except the *polyStippleEnable* variable is FALSE for the resulting objects being passed on to object setup. This will inhibit polygon stipple for these triangle objects.

```
TriangleFanDecomposition() {
        objectType = 3DOBJ_TRIANGLE
        nV = 3
        invertOrientation = FALSE
```

```
        provokingVtx = Triangle Fan Provoking Vertex Select
        if (primType == 3DPRIM_TRIFAN)
                polyStippleEnable = TRUE
        else  // primType == 3DPRIM_TRIFAN_NOSTIPPLE
                polyStippleEnable = FALSE
        v[0] arrays ← vIn[0] arrays      // the 1st vertex is common
        for each (vertex I in [1..vertexCount-2]) {
                v[1] arrays ← vIn[i] arrays
                v[2] arrays ← vIn[i+1] arrays
                ObjectSetup()
        }
}
```

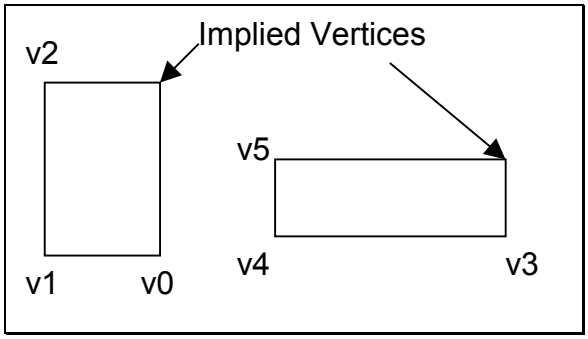
6.2.7 Polygon Decomposition

The 3DPRIM_POLYGON primitive is identical to the 3DPRIM_TRIFAN primitive with the exception that the *provokingVtx* is overridden with 0. This support has been added specifically for OpenGL support, avoiding the need for the driver to change the provoking vertex selection when switching between trifan and polygon primitives.

6.2.8 Rectangle List Decomposition

The 3DPRIM_RECTLIST primitive command specifies a list of independent, axis-aligned rectangles. Only the lower right, lower left, and upper left vertices (in that order) are included in the command – the upper right vertex is derived from the other vertices (in Object Setup).

Figure 6-7. 3DPRIM_RECTLIST Primitive



The decomposition of the 3DPRIM_RECTLIST primitive is identical to the 3DPRIM_TRILIST decomposition, with the exception of the *objectType* variable.

```
RectangleListDecomposition() {

        objectType = 3DOBJ_RECTANGLE

        nV = 3

        invertOrientation = FALSE

        provokingVtx = 0

        for each (vertex I in [0..vertexCount-3] by 3) {

                v[0] arrays ← vIn[i] arrays

                v[1] arrays ← vIn[i+1] arrays

                v[2] arrays ← vIn[i+2] arrays

                ObjectSetup()

        }

}
```

6.3 Object Setup

The Object Setup subfunction of the SF stage takes the post-viewport-transform data associated with each vertex of a basic object and computes various parameters required for scan conversion. This includes generation of implied vertices, translations and adjustments on vertex positions, and culling (removal) of certain classes of objects. The final object information is passed to the Windower/Masker (WM) stage where the object is rasterized into pixels.

6.3.1 Invalid Position Culling (Pre/Post-Transform)

At input the the SF stage, any objects containing a floating-point NaN value for Position X, Y, Z, or RHW will be unconditionally discarded. Note that this occurs on an object (not primitive) basis.

If Viewport Transformation is enabled, any objects containing a floating-point NaN value for post-transform Position X, Y or Z will be unconditionally discarded.

6.3.2 Viewport Transformation

If the **Viewport Transform Enable** bit of SF_STATE is ENABLED, a viewport transformation is applied to each vertex of the object.

The VPIndex associated with the leading vertex of the object is used to obtain the **Viewport Matrix Element** data from the corresponding element of the SF_VIEWPORT structure in memory. For each object vertex, the following scale and translate transformation is applied to the position coordinates:

x' = **m00** * x + **m30**

y' = **m11** * y + **m31**

z' = **m22** * z + **m32**

Software is responsible for computing the matrix elements from the viewport information provided to it from the API.

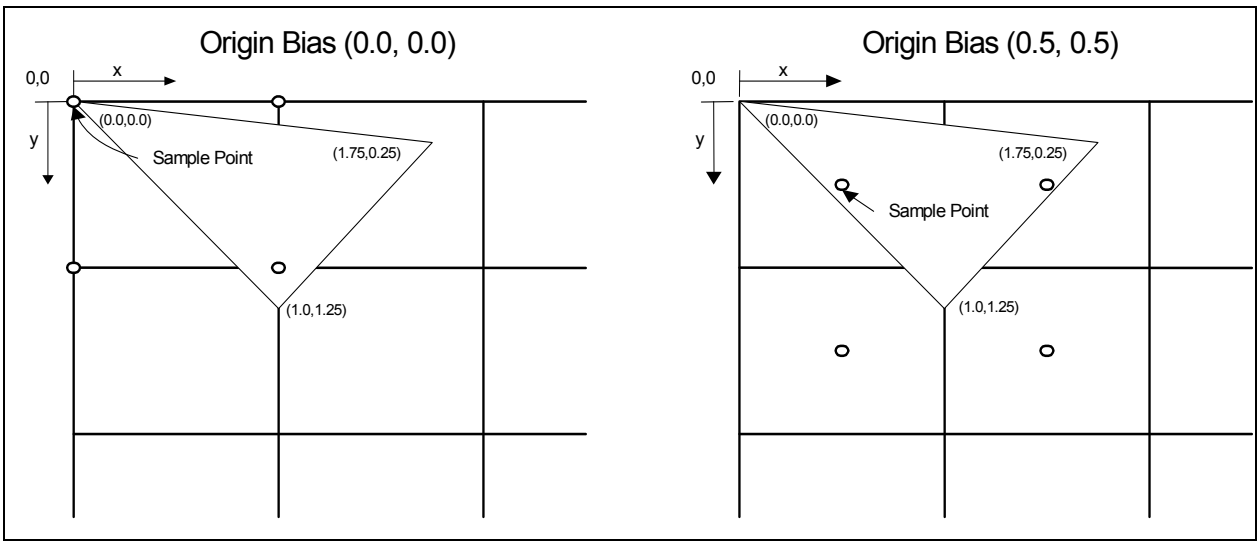
6.3.3 Destination Origin Bias

The positioning of the pixel sampling grid is programmable and is controlled by the **Destination Origin Horizontal/Vertical Bias** state variables (set via SF_STATE). If these bias values are both 0, pixels are sampled on an integer grid. Pixel (0,0) will be considered inside the object if the sample point at XY coordinate (0,0) falls within the primitive.

If the bias values are both 0.5, pixels are sampled on a "half" integer grid (i.e., X.5, Y.5). Pixel (0,0) will be considered inside the object if the sample point at XY coordinate (0.5,0.5) falls within the primitive. This positioning of the sample grid corresponds with the OpenGL rasterization rules, where "fragment centers" lay on a half-integer grid. It also corresponds with the Intel740 rasterizer (though that device did not employ "top left" rules).

Note that subsequent descriptions of rasterization rules for the various objects will be with reference to the pixel sampling grid.

Figure 6-8. Destination Origin Bias

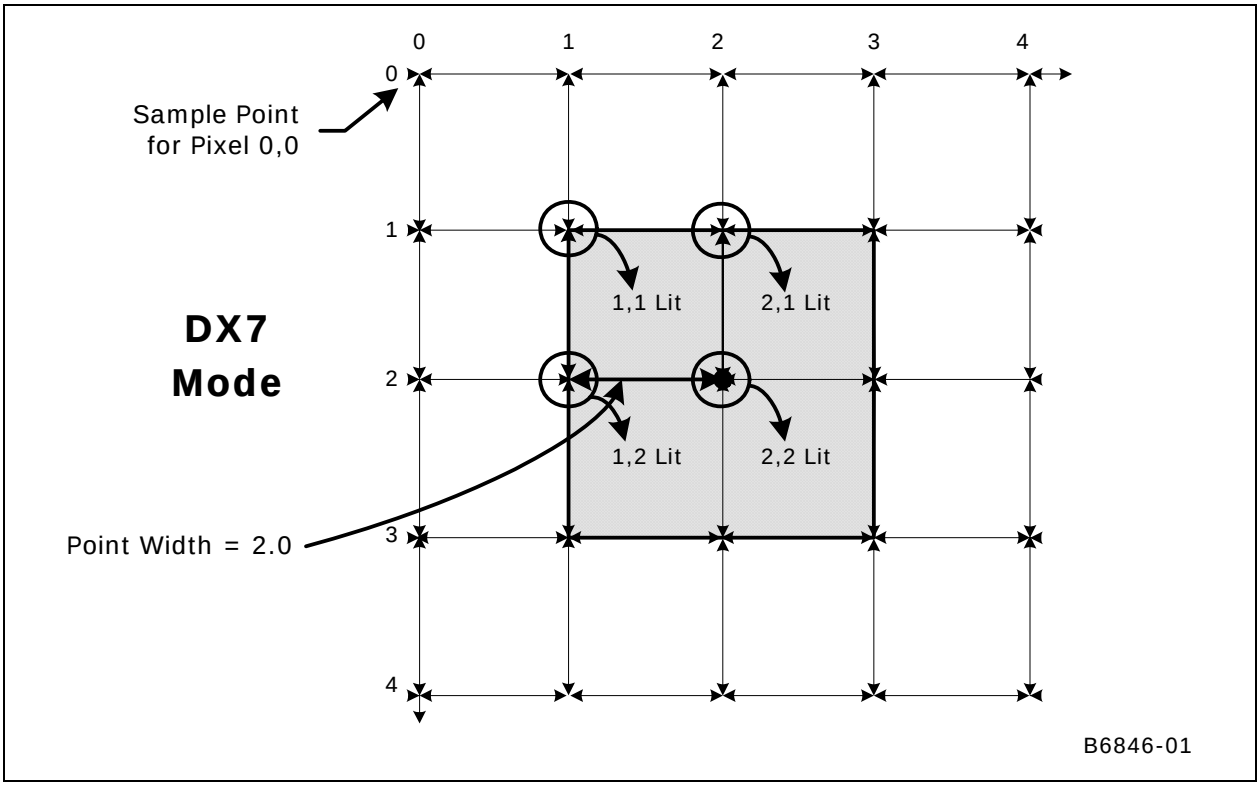


6.3.4 Point Rasterization Rule Adjustment

POINT objects are rasterized as square RECTANGLEs, with one exception: The **Point Rasterization Rule** state variable (in SF_STATE) controls the rendering of point object edges that fall directly on pixel sample points, as the treatment of these edge pixels varies between APIs.

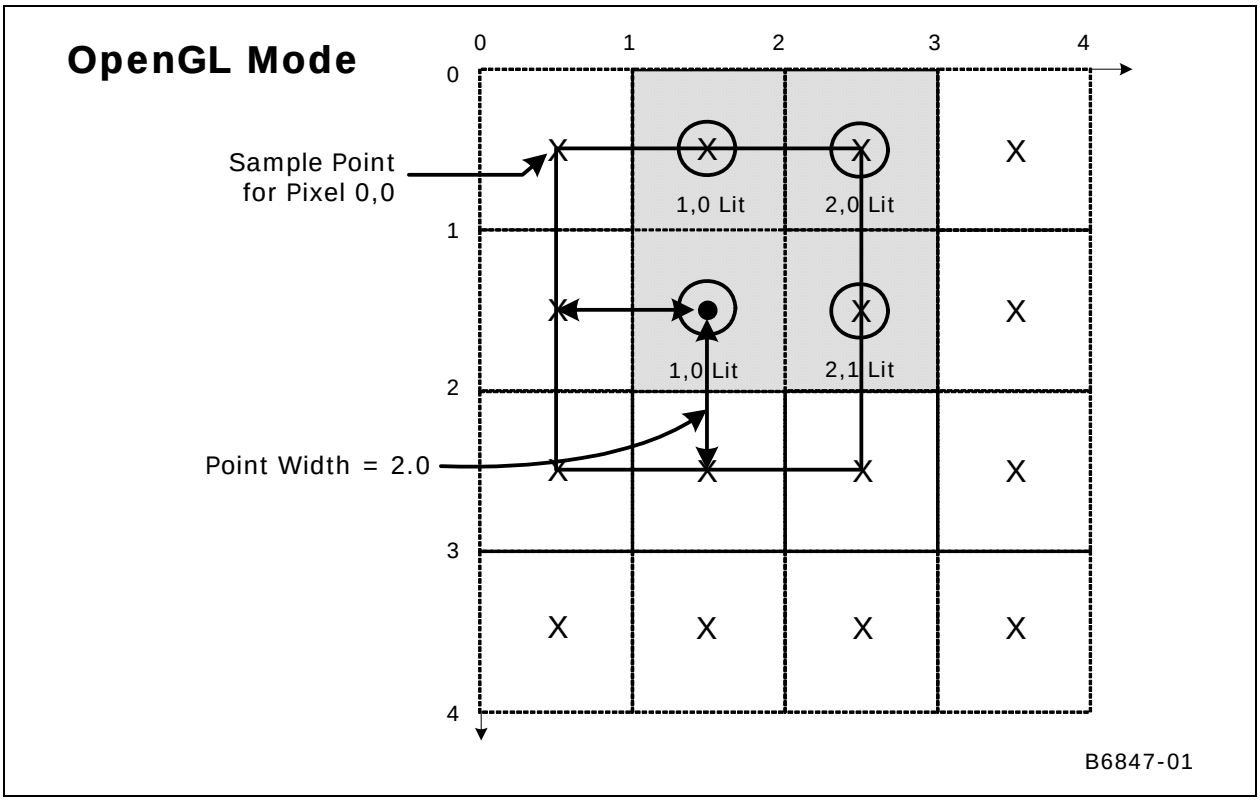
The following diagram shows the rasterization of a 2-pixel wide point centered at (2,2). Here the pixel sample grid coincides with the integer pixel coordinates, and the **Point Rasterization Rule** is set to RASTRULE_UPPER_LEFT. Note that the pixels which lie only on the upper and/or left edges are lit.

Figure 6-9. RASTRULE_UPPER_LEFT



The following diagram shows the rasterization of a 2-pixel wide point centered at (2,2) given “OpenGL” rasterization rules. Here the pixel sample grid coincides with half-integer pixel coordinates, and the **Point Rasterization Rule** is set to RASTRULE_UPPER_RIGHT. Note that the pixels which lie only on the upper and/or right edges are lit.

Figure 6-10. RASTRULE_UPPER_RIGHT



6.3.5 Drawing Rectangle Offset Application

The Drawing Rectangle Offset subfunction offsets the object's vertex X,Y positions by the pixel-exact, unclipped drawing rectangle origin (as programmed via the **Drawing Rectangle Origin X,Y** values in the 3DSTATE_DRAWING_RECTANGLE command). The Drawing Rectangle Offset subfunction (at least with respect to Color Buffer access) is unconditional, and therefore to (effectively) turn off the offset function the origin would need to be set to (0,0). A non-zero offset is typically specified when window-relative or viewport-relative screen coordinates are input to the device. Here the drawing rectangle origin would be loaded with the absolute screen coordinates of the window's or viewport's upper-left corner.

Clipping of objects which extend outside of the Drawing Rectangle occurs later in the pipeline. Note that this clipping is based on the "clipped" draw rectangle (as programmed via the **Clipped Drawing Rectangle** values in the 3DSTATE_DRAWING_RECTANGLE command), which must be clamped by software to the rendertarget boundaries. The unclipped drawing rectangle origin, however, can extend outside the screen limits in order to support windows whose origins are moved off-screen. This is illustrated in the following diagrams.

Figure 6-11. Onscreen Draw Rectangle

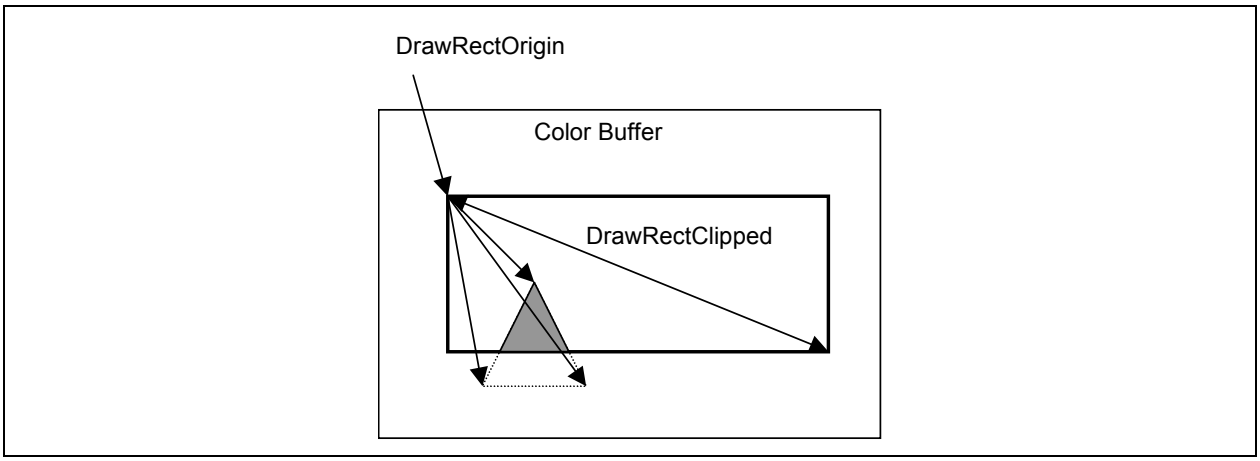
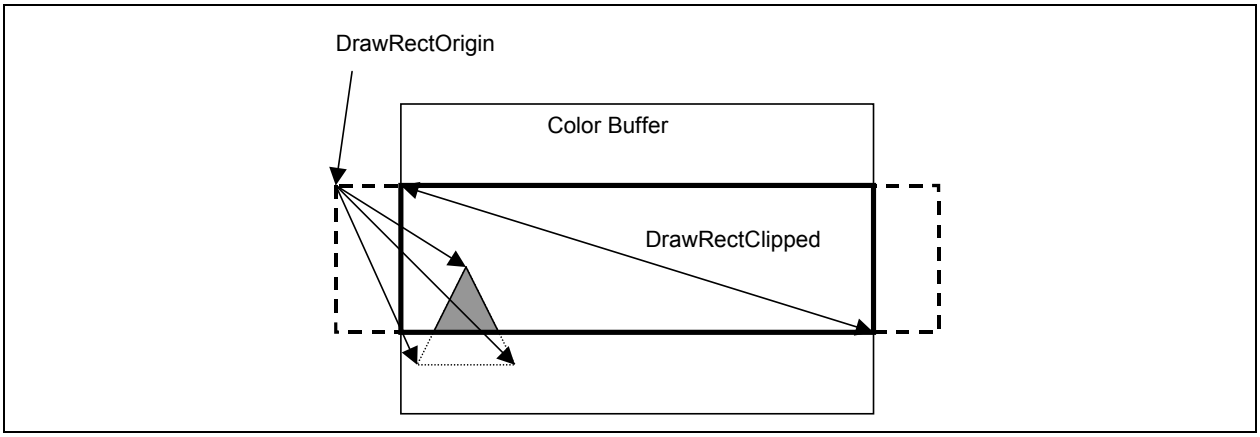


Figure 6-12. Partially-offscreen Draw Rectangle



6.3.5.1 3DSTATE_DRAWING_RECTANGLE

3DSTATE_DRAWING_RECTANGLE			
Project:	All	**Length Bias:**	2
The 3DSTATE_DRAWING_RECTANGLE command is used to set the 3D drawing rectangle and related state.			

DWord	Bit	Description
0	31:29	Command Type Default Value: 3h GFXPIPE Format: OpCode
	28:27	Command SubType Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	3D Command Opcode Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	3D Command Sub Opcode Default Value: 00h 3DSTATE_DRAWING_RECTANGLE Format: OpCode
	15:14	Reserved
	15:14	Reserved Project: DevSNB Format: MBZ
	7:0	DWord Length Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:16	Clipped Drawing Rectangle Y Min Project: All Format: U16 in Pixels from Color Buffer origin (upper left corner) FormatDesc Range [DevSNB]: [0,8191] (Device ignores bits 31:29) Specifies Ymin value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with Y coordinates *less than* Ymin will be clipped out. **Programming Notes** This value can be *larger than* **Clipped Drawing Rectangle Y Max.** If Ymin>Ymax, the clipped drawing rectangle is null, all polygons are discarded. If Ymin==Ymax, the clipped drawing rectangle is 1 pixel wide in the Y direction. **[DevSNB] Errata**: This field must be an even number

3DSTATE_DRAWING_RECTANGLE

	15:0	Clipped Drawing Rectangle X Min Project: All Format: U16 in Pixels from Color Buffer origin (upper left corner) FormatDesc Range [DevSNB]: [0,8191] (Device ignores bits 15:13) Specifies Xmin value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with X coordinates *less than* Xmin will be clipped out. **Programming Notes** — **Project** This value can be *larger than* **Clipped Drawing Rectangle X Max.** If Xmin>Xmax, the clipped drawing rectangle is null, all polygons are discarded. If Xmin==Xmax, the clipped drawing rectangle is 1 pixel wide in the X direction. — All
2	31:16	Clipped Drawing Rectangle Y Max Project: All Format: U16 in Pixels from Color Buffer origin (upper left corner) FormatDesc Range [DevSNB]: [0,8191] (Device ignores bits 31:29) Specifies Ymax value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with coordinates *greater than* Ymax will be clipped out. **Programming Notes** This value can be *less than* **Clipped Drawing Rectangle Y Min.** If Ymax<Ymin, the clipped drawing rectangle is null, all polygons are discarded. If Ymin==Ymax, the clipped drawing rectangle is 1 pixel wide in the Y direction.
	15:0	Clipped Drawing Rectangle X Max Project: All Format: U16 in Pixels from Color Buffer origin (upper left corner) FormatDesc Range [DevSNB]: [0,8191] (Device ignores bits 15:13) Specifies Xmax value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with coordinates *greater than* Xmax will be clipped out. **Programming Notes** — **Project** This value can be less *than* **Clipped Drawing Rectangle X Min.** If Xmax<Xmin, the clipped drawing rectangle is null, all polygons are discarded.If Xmin==Xmax, the clipped drawing rectangle is 1 pixel wide in the X direction. — All

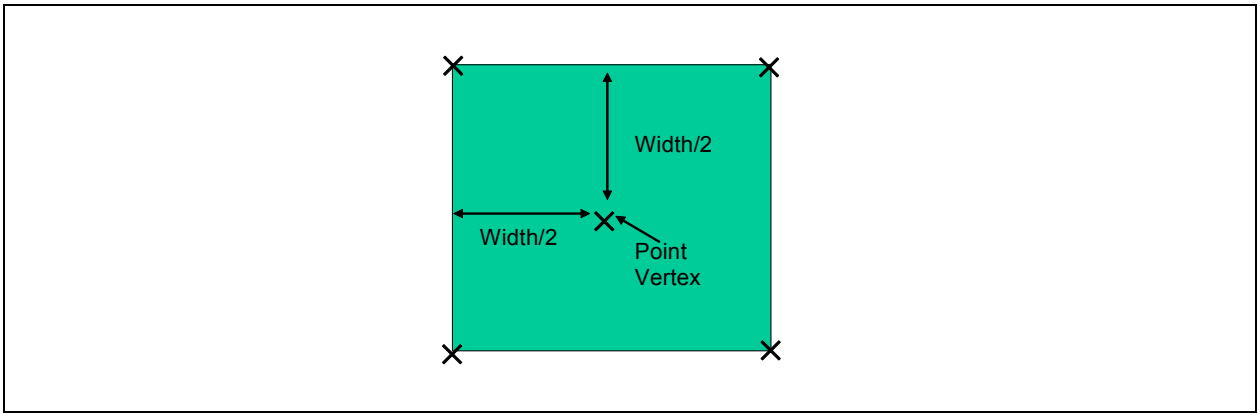
3DSTATE_DRAWING_RECTANGLE		
3	31:16	Drawing Rectangle Origin Y Project: All Format: S15 in Pixels from Color Buffer origin (upper left corner). FormatDesc Range [DevSNB]: [-8192,8191] (Bits 31:30 should be a sign extension) Specifies Y origin of Drawing Rectangle (in whole pixels) relative to origin of the Color Buffer, used to map incoming (Draw Rectangle-relative) vertex positions to the Color Buffer space.
	15:0	Drawing Rectangle Origin X Project: All Format: S15 in Pixels from Color Buffer origin (upper left corner). FormatDesc Range [DevSNB]: [-8192,8191] (Bits 31:30 should be a sign extension) Specifies X origin of Drawing Rectangle (in whole pixels) relative to origin of the Color Buffer, used to map incoming (Draw Rectangle-relative) vertex positions to the Color Buffer space.

6.3.6 Point Width Application

This stage of the pipeline applies only to 3DOBJ_POINT objects. Here the point object is converted from a single vertex to four vertices located at the corners of a square centered at the point's X,Y position. The width and height of the square are specified by a *point width* parameter. The **Use Point Width State** value in SF_STATE determines the source of the point width parameter: the point width is either taken from the **Point Width** value programmed in SF_STATE or the PointWidth specified with the vertex (as read back from the vertex VUE earlier in the pipeline).

The corner vertices are computed by adding and subtracting one half of the point width, as shown in Figure 6-13.

Figure 6-13. Point Width Application

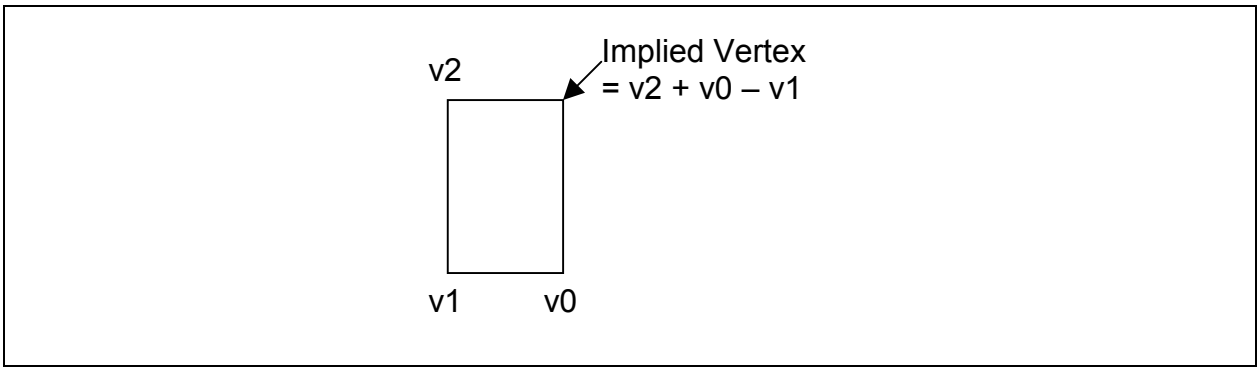


Z and W vertex attributes are copied from the single point center vertex to each of the four corner vertices.

6.3.7 Rectangle Completion

This stage of the pipeline applies only to 3DOBJ_RECTANGLE objects. Here the X,Y coordinates of the 4th (upper right) vertex of the rectangle object is computed from the first 3 vertices as shown in the following diagram. The other vertex attributes assigned to the implied vertex (v[3]) are UNDEFINED as they are not used. The Object Setup subfunction will use the values at only the first 3 vertices to compute attribute interpolants used across the entire rectangle.

Figure 6-14. Rectangle Completion



6.3.8 Vertex X,Y Clamping and Quantization

At this stage of the pipeline, vertex X and Y positions are in continuous screen (pixel) coordinates. These positions are quantized to subpixel precision by rounding the incoming values to the nearest subpixel (using round-to-nearest-or-even rules). The device supports rasterization with either 4 or 8 fractional (subpixel) position bits, as specified by the **Vertex SubPixel Precision Select** bit of SF_STATE.

The vertex X and Y screenspace coordinates are also **clamped** to the fixed-point "guardband" range supported by the rasterization hardware, as listed in the following table:

Table 19 Per-Device Guardband Extents

Device	Supported X,Y ScreenSpace "Guardband" Extent	Maximum Post-Clamp Delta (X or Y)
DevSNB	[-16K,16K-1]	16K

For earlier releases, an additional restriction effectively cuts the guardband extent in half: The screen-aligned 2D bounding-box of an object must not exceed 8K pixels in either X or Y. E.g., a line between (-6K,-6K) and (6K,6K) would not be rendered correctly, as its bounding box is 12K pixels in X and Y. This restriction effectively requires software to ensure all objects are contained within, or clipped to, a 2D region not exceeding 8K pixels in X or Y (even though that region can be located anywhere within the [-8K,8K-1] guardband extent). A similar restriction applies to [DevSNB], though the guardband and maximum delta are doubled from legacy products.

Note that this clamping occurs after the Drawing Rectangle Origin has been applied and objects have been expanded (i.e., points have been expanded to squares, etc.). In almost all circumstances, if an object's vertices are actually modified by this clamping (i.e., had X or Y coordinates outside of the guardband extent the rendered object will not match the intended result. Therefore software should take steps to ensure that this does not happen – e.g., by clipping objects such that they do not exceed these limits after the Drawing Rectangle is applied.

In addition, in order to be correctly rendered, objects must have a screenspace bounding box not exceeding 8K in the X or Y direction. This additional restriction must also be comprehended by software, i.e., enforced by use of clipping.

6.3.9 Degenerate Object Culling

At this stage of the pipeline, “degenerate” objects are discarded. This operation is automatic and cannot be disabled. (The object rasterization rules would by definition cause these objects to be “invisible” – this culling operation is mentioned here to reinforce that the device implementation optimizes these degeneracies as early as possible).

Degenerate objects are defined in the following table.

Table 20. Degenerate Objects

objType	Degenerate Object Definition
3DOBJ_POINT	Two or more corner vertices are coincident (i.e., the radius quantized to zero)
3DOBJ_LINE	The endpoints are coincident
3DOBJ_TRIANGLE	All three vertices are collinear or any two vertices are coincident and SOLID fill mode applies to the triangle
3DOBJ_RECTANGLE	Two or more corner vertices are coincident

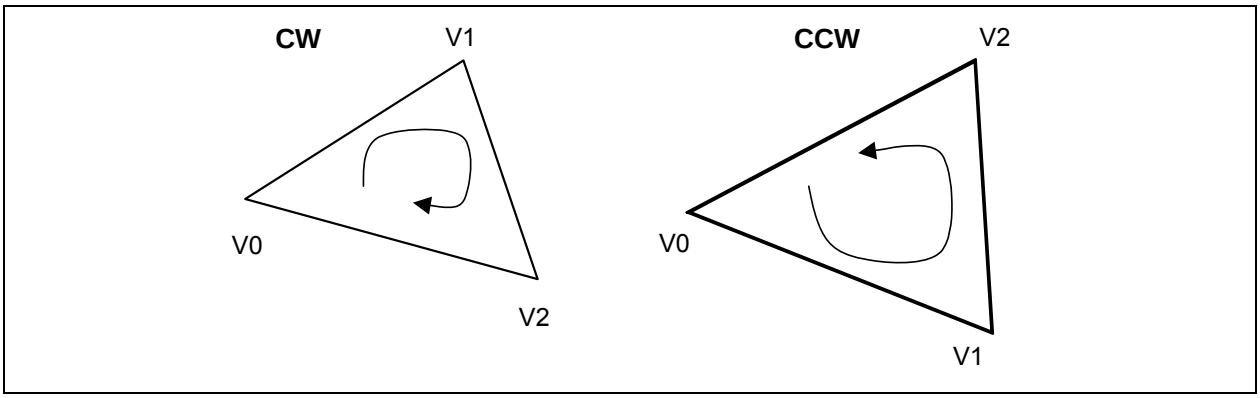
6.3.10 Triangle Orientation (Face) Culling

At this stage of the pipeline, 3DOBJ_TRIANGLE objects can be optionally discarded based on the “face orientation” of the object. This culling operation does not apply to the other object types.

This operation is typically called “back face culling”, though front facing objects (or all 3DOBJ_TRIANGLE objects) can be selected to be discarded as well. Face culling is typically used to eliminate triangles facing away from the viewer, thus reducing rendering time.

The “winding order” of a triangle is defined by the the triangle vertex’s 2D (X,Y) screen space position when traversed from v0 to v1 to v2. That traversal will proceed in either a clockwise (CW) or counter-clockwise (CCW) direction, as shown in the following figure. A degenerate triangle is considered “backfacing”, regardless of the FrontWinding state.)

Figure 6-15. Triangle Winding Order



The **Front Winding** state variable in SF_STATE controls whether CW or CCW triangles are considered as having a "front-facing" orientation (at which point non-front-facing triangles are considered "back-facing"). The internal variable *invertOrientation* associated with the triangle object is then used to determine whether the orientation of a that triangle should be inverted. Recall that this variable is set in the Primitive Decomposition stage to account for the alternating orientations of triangles in strip primitives resulting form the ordering of the vertices used to process them.

The **Cull Mode** state variable in SF_STATE specifies how triangles are to be discarded according to their resultant orientation, as defined in Table 21.

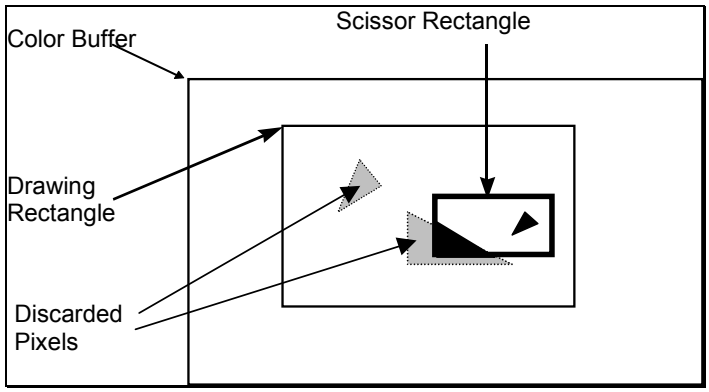
Table 21. Cull Mode

CullMode	Definition
CULLMODE_NONE	The face culling operation is disabled
CULLMODE_FRONT	Triangles with "front facing" orientation are discarded
CULLMODE_BACK	Triangles with "back facing" orientation are discarded
CULLMODE_BOTH	All triangles are discarded

6.3.11 Scissor Rectangle Clipping

A *scissor* operation can be used to restrict the extent of rendered pixels to a screen-space aligned rectangle. If the scissor operation is enabled, portions of objects falling outside of the intersection of the scissor rectangle and the clipped draw rectangle are clipped (pixels discarded).

The scissor operation is enabled by the **Scissor Rectangle Enable** state variable in SF_STATE. If enabled, the VPIndex associated with the leading vertex of the object is used to select the corresponding SF_VIEWPORT structure. Up to 16 structures are supported. The **Scissor Rectangle X,Y Min,Max** fields of the SF_VIEWPORT structure defines a scissor rectangle as a rectangle in integer pixel coordinates relative to the (unclipped) origin of the Drawing Rectangle. The scissor rectangle is defined relative to the Drawing Rectangle to better support the OpenGL API. (OpenGL specifies the "Scissor Box" in window-relative coordinates). This allows instruction buffers with embedded Scissor Rectangle definitions to remain valid even after the destination window (drawing rectangle) moves.



Specifying either scissor rectangle xmin > xmax or ymin > ymax will cause all polygons to be discarded for a given viewport (effectively a null scissor rectangle).

6.3.12 Line Rasterization

The device supports three styles of line rendering: *zero-width* (*cosmetic*) lines, *non-antialiased* lines, and *antialiased* lines. Zero-Width lines are rendered according to the Grid Intersection Quantization (GIQ) technique. Non-antialiased lines are rendered as a polygon having a specified width as measured parallel to the major axis of the line. Antialiased lines are rendered as a rectangle having a specified width measured perpendicular to the line connecting the vertices.

The functions required to render lines is split between the SF and WM units. The SF unit is responsible for computing the overall geometry of the object to be rendered, including the pixel-exact bounding box, edge equations, etc., and therefore is provided with the screen-geometry-related state variables. The WM unit performs the actual scan conversion, determining the exact pixel included/excluded and coverage value for anti-aliased lines.

6.3.12.1 Zero-Width (Cosmetic) Line Rasterization

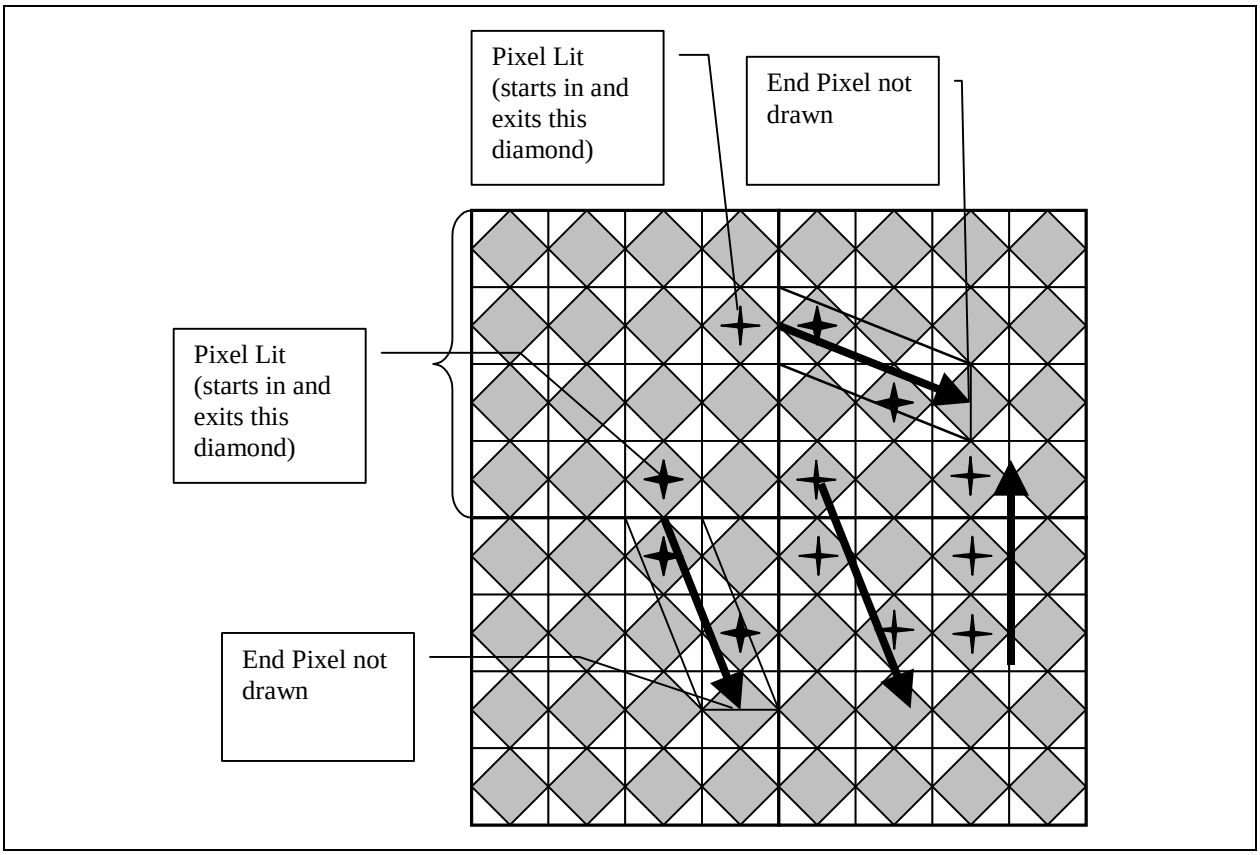
(The specification of zero-width line rasterization would be more correctly included in the WM Unit chapter, though is being included here to keep it with the rasterization details of the other line types).

When the **Line Width** is set to zero, the device will use special rules to rasterize zero-width ("cosmetic") lines. The **Anti-Aliasing Enable** state variable is ignored when **Line Width** is zero.

When the *LineWidth* is set to zero, the device will use special rules to rasterize "cosmetic" lines. The rasterization rules used are compliant with the *Grid Intersection Quantization (GIQ)* (aka diamond exit rules) algorithm. The rasterization rules comply with the OpenGL conformance requirements (for 1-pixel wide non-smooth lines).

The GIQ rules basically intersect the directed, ideal line connecting two endpoints with an array of diamond-shaped areas surrounding pixel sample points. Wherever the line exits a diamond (including passing through a diamond), the corresponding pixel is lit. Special rules are used to define the subpixel locations which are considered interior to the diamonds, as a function of the slope of the line. When a line ends in a diamond (and therefore does not exit that diamond), the corresponding pixel is not drawn. When a line starts in a diamond and exits that diamond, the corresponding pixel is drawn.

The following diagram shows some examples of GIQ-rendered lines.

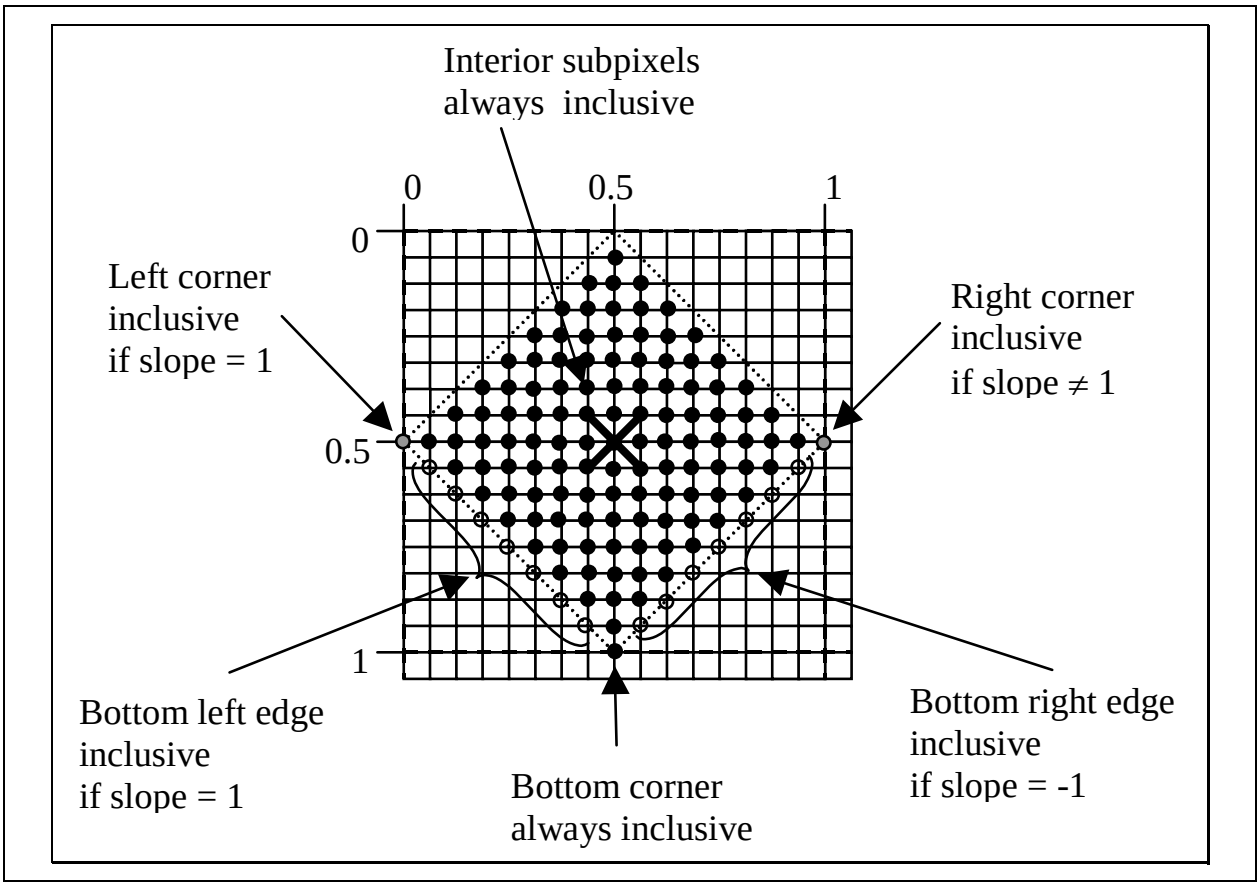


The following subsections describe the GIQ rules in more detail.

6.3.12.2 GIQ (Diamond) Sampling Rules – Legacy Mode

When the **Legacy Line Rasterization Enable** bit in WM_STATE is ENABLED, zero-width lines are rasterized according to the algorithm presented in this subsection. Also note that the **Last Pixel Enable** bit of SF_STATE controls whether the last pixel of the last line in a LINESTRIP_xxx primitive or the last pixel of each line in a LINELIST_xxx primitive is rendered.

Refer to the following figure, which shows the neighborhood of subpixels around a given pixel sample point. Note that the device divides a pixel into a 16x16 array of subpixels, referenced by their upper left corners.



The solid-colored subpixels are considered "interior" to the diamond centered on the pixel sample point. Here the Manhattan distance to the pixel sample point (center) is less than ½.

The subpixels falling on the edges of the diamond (Manhattan distance = ½) are exclusive, with the following exceptions:

1. **The bottom corner subpixel is always inclusive.** This is to ensure that lines with slopes in the open range (-1,1) touch a diamond even when they cross exactly between pixel diamonds.
2. **The right corner subpixel is inclusive as long as the line slope is not exactly one, in which case the left corner subpixel is inclusive.** Including the right corner subpixel ensures that lines with slopes in the range (1, +infinity] or [-infinity, -1) touch a diamond even when they cross exactly between pixel diamonds. Including the left corner on slope=1 lines is required for proper handling of slope=1 lines (see (3) below) – where if the right corner was inclusive, a slope=1 line falling exactly between pixel centers would wind up lighting pixel on both sides of the line (not desired).
3. **The subpixels along the bottom left edge are inclusive only if the line slope = 1.** This is to correctly handle the case where a slope=1 line falls enters the diamond through a left or bottom corner and ends on the bottom left edge. One does not consider this "passing through" the diamond (where the normal rules would have us light the pixel). This is to avoid the following case: One slope=1 line segment enters through one corner and ends on the edge, and another (continuation) line segments starts at that point on the edge and exits through the other corner. If simply passing through a corner caused the pixel to be lit, this case would case the pixel to be lit twice – breaking the rule that connected line segments should not cause double-hits or missing

pixels. So, by considering the entire bottom left edge as “inside” for slope=1 lines, we will only light the pixel when a line passes through the entire edge, or starts on the edge (or the left or bottom corner) and exits the diamond.

4. **The subpixels along the bottom right edge are inclusive only if the line slope = -1.** Similar case as (3), except slope=-1 lines require the bottom right edge to be considered inclusive.

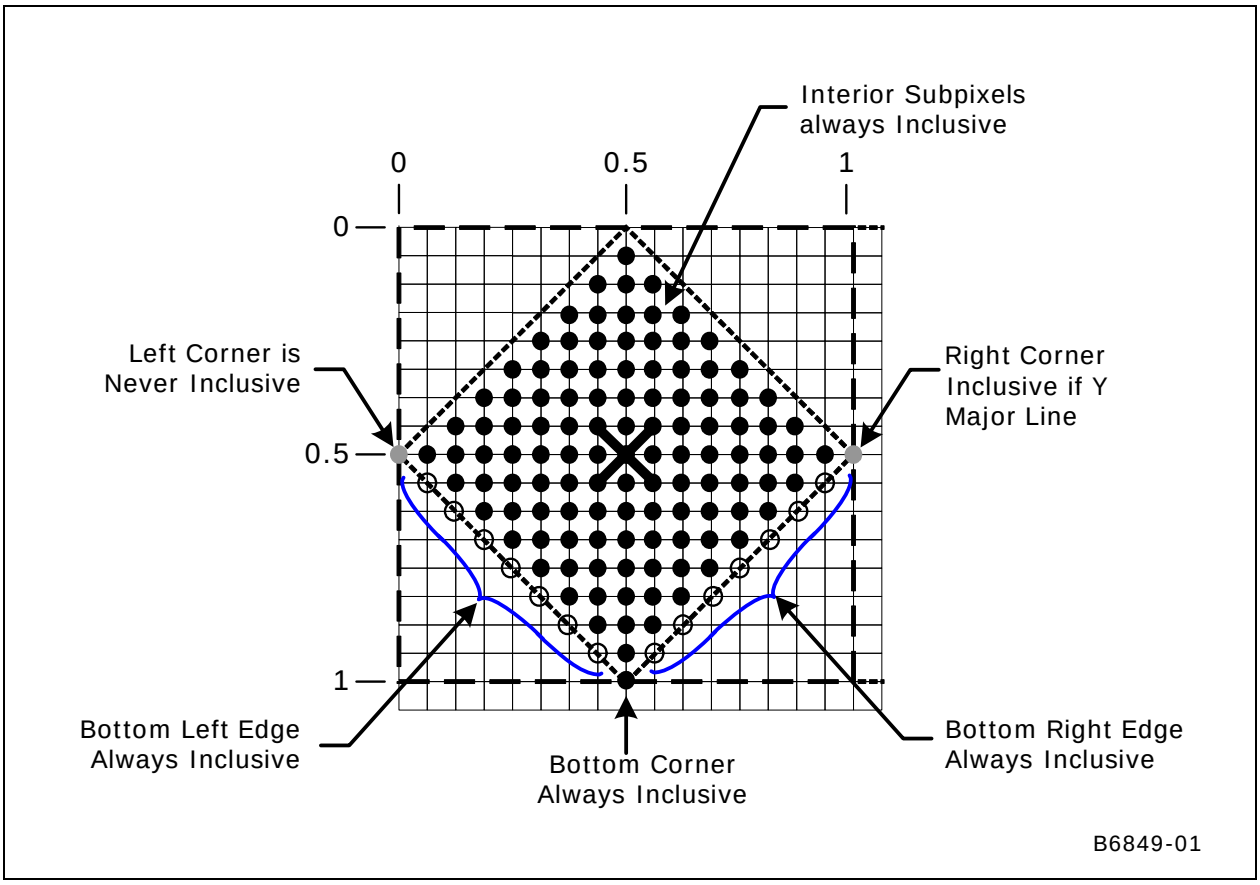
The following equation determines whether a point (point.x, point.y) is inside the diamond of the pixel sample point (sample.x, sample.y), given additional information about the slope (slopePosOne, slopeNegOne).

```
delta_x            = point.x – sample.x

delta_y            = point.y – sample.y

distance           = abs(delta_x) + abs(delta_y)

interior           = (distance < 0.5)

bottom_corner      = (delta_x == 0.0)   && (delta_y == 0.5)

left_corner        = (delta_x == –0.5) && (delta_y == 0.0)

right_corner       = (delta_x == 0.5)   && (delta_y == 0.0)

bottom_left_edge   = (distance == 0.5) && (delta_x < 0) && (delta_y > 0)

bottom_right_edge  = (distance == 0.5) && (delta_x > 0) && (delta_y > 0)

inside = interior ||

         bottom_corner ||

         (slopePosOne ? left_corner : right_corner) ||

         (slopePosOne && left_edge) ||

         (slopeNegOne && right_edge)
```

6.3.12.3 GIQ (Diamond) Sampling Rules

When the **Legacy Line Rasterization Enable** bit in WM_STATE is DISABLED, zero-width lines are rasterized according to the algorithm presented in this subsection. Also note that the **Last Pixel Enable** bit of SF_STATE controls whether the last pixel of the last line in a LINESTRIP_xxx primitive or the last pixel of each line in a LINELIST_xxx primitive is rendered.

Refer to the following figure, which shows the neighborhood of subpixels around a given pixel sample point. Note that the device divides a pixel into a 16x16 array of subpixels, referenced by their upper left corners.



The solid-colored subpixels are considered “interior” to the diamond centered on the pixel sample point. Here the Manhattan distance to the pixel sample point (center) is less than ½.

The subpixels falling on the edges of the diamond (Manhattan distance = ½) are exclusive, with the following exceptions:

1. **The bottom corner subpixel is always inclusive.** This is to ensure that lines with slopes in the open range (-1,1) touch a diamond even when they cross exactly between pixel diamonds.

2. **The right corner subpixel is inclusive as long as the line is not X Major (X Major is defined as -1 <= slope <= 1).** Including the right corner subpixel ensures that lines with slopes in the range (>1, +infinity] or [-infinity, <-1) touch a diamond even when they cross exactly between pixel diamonds.

3. **The left corner subpixel is never inclusive.** For Y Major lines, having the right corner subpixel as always inclusive requires that the left corner subpixel should never be inclusive, since a line falling exactly between pixel centers would wind up lighting pixel on both sides of the line (not desired).

4. **The subpixels along the bottom left edge are always inclusive.** This is to correctly handle the case where a line enters the diamond through a left or bottom corner and ends on the bottom left edge. One does not consider this “passing through” the diamond (where the normal rules would have us light the pixel). This is to avoid the following case: One line segment enters through one corner and ends on the edge, and another (continuation) line segments starts at that point on the edge and exits through the other corner. If simply passing through a corner caused the pixel to be lit, this case would case the pixel to be lit twice – breaking the rule that connected line segments

should not cause double-hits or missing pixels. So, by considering the entire bottom left edge as "inside", we will only light the pixel when a line passes through the entire edge, or starts on the edge (or the left or bottom corner) and exits the diamond.

5. **The subpixels along the bottom right edge are always inclusive.** Same as case as (4), except slope=-1 lines require the bottom right edge to be considered inclusive.

The following equation determines whether a point (point.x, point.y) is inside the diamond of the pixel sample point (sample.x, sample.y), given additional information about the slope (XMajor).

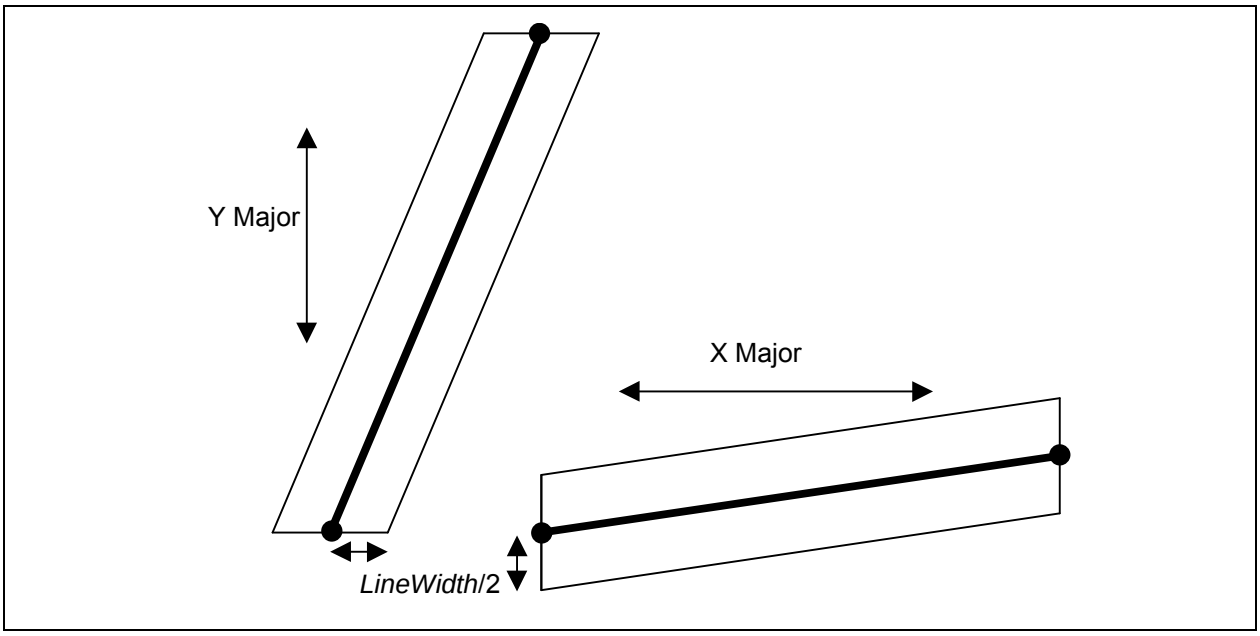
```
delta_x          = point.x – sample.x

delta_y          = point.y – sample.y

distance         = abs(delta_x) + abs(delta_y)

interior         = (distance < 0.5)

bottom_corner    = (delta_x == 0.0)   && (delta_y == 0.5)

left_corner      = (delta_x == –0.5) && (delta_y == 0.0)

right_corner     = (delta_x == 0.5)   && (delta_y == 0.0)

bottom_left_edge = (distance == 0.5) && (delta_x < 0) && (delta_y > 0)

bottom_right_edge = (distance == 0.5) && (delta_x > 0) && (delta_y > 0)

inside = interior ||

         bottom_corner ||

         (!XMajor && right_corner) ||

         ( bottom_left_edge) ||

         ( bottom_right_edge)
```

6.3.12.4 Non-Antialiased Wide Line Rasterization

Non-anti-aliased, non-zero-width lines are rendered as parallelograms that are centered on, and aligned to, the line joining the endpoint vertices. Pixels sampled interior to the parallelogram are rendered; pixels sampled exactly on the parallelogram edges are rendered according to the polygon "top left" rules.

The parallelogram is formed by first determining the major axis of the line (diagonal lines are considered x-major). The corners of the parallelogram are computed by translating the line endpoints by +/-(**Line Width** / 2) in the direction of the minor axis, as shown in the following diagram.

Figure 6-16. Non-Antialiased Line Rasterization

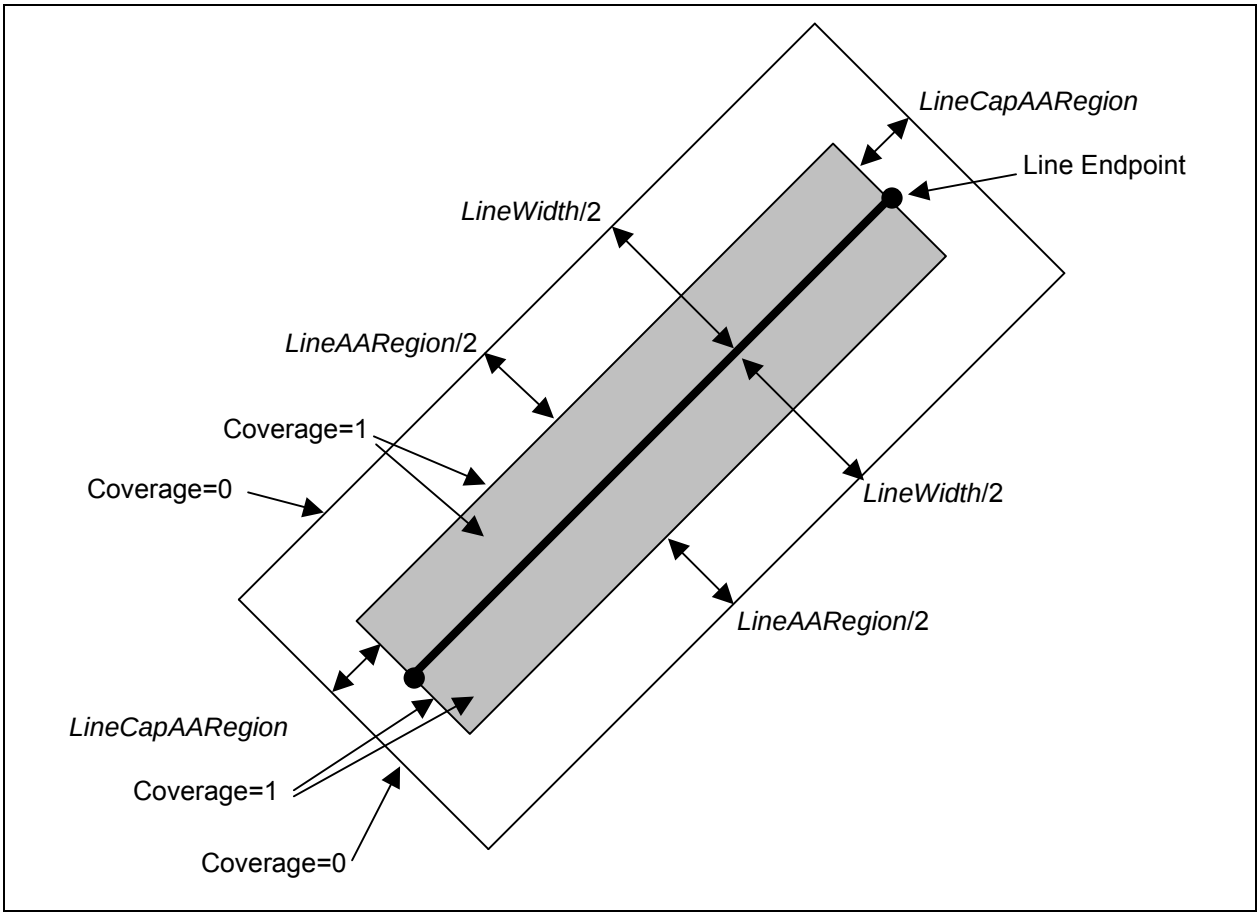


6.3.12.5 Anti-aliased Line Rasterization

Anti-aliased lines are rendered as rectangles that are centered on, and aligned to, the line joining the endpoint vertices. For each pixel in the rectangle, a fractional coverage value (referred to as Antialias Alpha) is computed – this coverage value will normally be used to attenuate the pixel's alpha in the pixel shader thread. The resultant alpha value is therefore available for use in those downstream pixel pipeline stages in order to generate the desired effect (e.g., use the attenuated alpha value to modulate the pixel's color, and add the result to the destination color, etc.). Note that software is required to explicitly program the pixel shader and pixel pipeline to obtain the desired anti-aliasing effect – the device will simply make the coverage-attenuated pixel alpha values available for use in the pixel shader.

The dimensions of the rendered rectangle, and the parameters controlling the coverage value computation, are programmed via the **Line Width**, **Line AA Region**, and **Line Cap AA Region** state variables, as shown below. The edges parallel to the line are located at the distance (*LineWidth*/2) from the line (measured in screen pixel units perpendicular to the line). The end-cap edges are perpendicular to the line and located at the distance (*LineCapAARegion*) from the endpoints.

Figure 6-17. Anti-aliased Line Rasterization



Along the parallel edges, the coverage values ramp from the value 0 at the very edges of the rectangle to the value 1 at the perpendicular distance (*LineAARegion*/2) from a given edge (in the direction of the line). A pixel's coverage value is computed with respect to the closest edge. In the cases where (*LineAARegion*/2) < (*LineWidth*/2), this results in a region of fractional coverage values near the edges of the rectangle, and a region of "fully-covered" coverage values (i.e., the value 1) at the interior of the line. When (*LineAARegion*/2) == (*LineWidth*/2), only pixel sample points falling exactly on the line can generate fully-covered coverage values. If (*LineAARegion*/2) > (*LineWidth*/2), no pixels can be fully-covered (it is expected that this case is not typically desired).

Along the end cap edges, the coverage values ramp from the value 1 at the line endpoint to the value 0 at the cap edge – itself at a perpendicular distance (*LineCapAARegion*) from the endpoint. Note that, unlike the line-parallel edges, there is only a single parameter (*LineCapAARegion*) controlling the extension of the line at the end caps and the associated coverage ramp.

The regions near the corners of the rectangle have coverage values influenced by distances from both the line-parallel and end cap edges – here the two coverage values are multiplied together to provide a composite coverage value.

The computed coverage value for each pixel is passed through the Windower Thread Dispatch payload. The Pixel Shader kernel should be passed (unmodified) by the shader to the Render Cache as part of it's output message.

6.3.12.5.1 Anti-aliased Line Distance Mode

In legacy devices, the distance from a pixel to the line is approximated by the “Manhattan Distance” (abs(delta_x)+abs)delta_y). More recently, a better approximation to the true perpendicular distance has been added for better visual quality and API compliance. On those devices, the **AA Line Distance Mode** bit in SF_STATE can be used to select between the legacy and improved distance calculations.

6.4 SF Pipeline State Summary

6.4.1 3DSTATE_SF [DevSNB+]

6.4.1.1 3DSTATE_SF [DevSNB]

For **[DevSNB]**, the state used by the SF stage is defined with this inline state packet.

3DSTATE_SF				
Project:	[DevSNB]		**Length Bias:**	2
DWord	**Bit**	**Description**		
0	31:29	Command Type Default Value: 3h GFXPIPE Format: OpCode		
	28:27	Command SubType Default Value: 3h GFXPIPE_3D Format: OpCode		
	26:24	3D Command Opcode Default Value: 0h 3DSTATE_PIPELINED Format: OpCode		
	23:16	3D Command Sub Opcode Default Value: 13h 3DSTATE_SF Format: OpCode		
	15:8	Reserved Project: All Format: MBZ		
	7:0	DWord Length Default Value: 12h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All		
1	31:28	Reserved Project: All Format: MBZ		

3DSTATE_SF		
	27:22	Number of SF Output Attributes Project: All Format: U6 Count of attributes Range [0,48] Specifies the number of vertex attributes passed from the SF stage to the WM stage (does not include Position). The actual number of attributes specified by this field must be set the same as the **Number of SF Output Attributes** field in 3DSTATE_WM. In the range description below, “swizzling” refers to the operations controlled by the following state fields: Attribute n Component Override X/Y/Z/W Attribute n Constant Source Attribute n Swizzle Select Attribute n Source Attribute Attribute n WrapShortest Enables 0: Specifies no attributes (beyond position) are associated with vertices. 1-16: Specifies 1-16 attributes. Swizzling performed on Attributes 0-15 (as required). 17-32: Specifies 17-32 attributes. Swizzling performed on Attributes 0-15. Attributes 16-31 (as required) passed through unmodified. 33-48: Specifies 17-32 attributes (# attributes = field value – 16). Swizzling performed on Attributes 16-31 (as required) only. Attributes 0-15 passed through unmodified. **Note**: Attribute n Component Override and Constant Source states apply to Attributes 16-31 (as required) instead of Attributes 0-15. E.g., this allows an Attribute 16-31 component to be overridden with the PrimitiveID value. Attribute n WrapShortest Enables still apply to Attributes 0-15. Attribute n Swizzle Select and Attribute n Source Attribute states are ignored and none of the swizzling functions available through these controls are performed.

	21	Attribute Swizzle Enable Project: All Format: Enable FormatDesc This bit controls the use of the **Attribute n Swizzle Select** and **Attribute n Source Attribute** fields only. If ENABLED, those fields are used as described below. If DISABLED, attributes are copied from their corresponding source attributes, for the purposes of Swizzle Select only. Note that the following fields are unaffected by this bit, and are therefore always used to control their respective fields: Attribute n Component Override X/Y/Z/W Attribute n Constant Source Attribute n WrapShortest Enables See Number of SF Output Attributes field.
	20	Point Sprite Texture Coordinate Origin Project: All Format: U1 enumerated type FormatDesc This state controls how Point Sprite Texture Coordinates are generated (when enabled on a per-attribute basis by **Point Sprite Texture Coordinate Enable**).
	19:16	Reserved Project: All Format: MBZ

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

3DSTATE_SF

	15:11	Vertex URB Entry Read Length Project: All Format: U5 FormatDesc Range [1,16] Specifies the amount of URB data read for each Vertex URB entry, in 256-bit register increments. **Programming Notes** (Project) — All It is UNDEFINED to set this field to 0 indicating no Vertex URB data to be read. This field should be set to the minimum length required to read the maximum source attribute. The maximum source attribute is indicated by the maximum value of the enabled **Attribute # Source Attribute** if **Attribute Swizzle Enable** is set, **Number of Output Attributes**-1 if enable is not set. read_length = ceiling((max_source_attr+1)/2) [errata] Corruption/Hang possible if length programmed larger than recommended
	10	Reserved Project: All Format: MBZ
	9:4	Vertex URB Entry Read Offset Project: All Format: U6 FormatDesc Range [0,63] Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB.
	3:0	Reserved Project: All Format: MBZ
2	31:12	Reserved Project: All Format: MBZ
	11	Legacy Global Depth Bias Enable Project: All Format: Enable FormatDesc Enables the SF to use the Global Depth Offset Constant state unmodified. If this bit is not set, the SF will scale the Global Depth Offset Constant.
	10	Statistics Enable Project: All Format: Enable FormatDesc If ENABLED, this FF unit will increment CL_PRIMITIVES_COUNT on behalf of the CLIP stage. If DISABLED, CL_PRIMITIVES_COUNT will be left unchanged. **Programming Notes** (Project) — All This bit should be set whenever clipping is enabled and the **Statistics Enable** bit is set in CLIP_STATE. It should be cleared if clipping is disabled *or* **Statistics Enable** in CLIP_STATE is clear.

	9	Global Depth Offset Enable Solid Project: All Format: Enable FormatDesc Enables computation and application of Global Depth Offset for SOLID objects.
	8	Global Depth Offset Enable Wireframe Project: All Format: Enable FormatDesc Enables computation and application of Global Depth Offset when triangles are rendered in WIREFRAME mode.
	7	Global Depth Offset Enable Point Project: All Format: Enable FormatDesc Enables computation and application of Global Depth Offset when triangles are rendered in POINT mode.
	6:5	FrontFace Fill Mode Project: All Format: U2 enumerated type FormatDesc This state controls how front-facing triangle and rectangle objects are rendered.

Value	Name	Description	Project
0h	SOLID	Any triangle or rectangle object found to be front-facing is rendered as a solid object. This setting is required when rendering rectangle (RECTLIST) objects.	All
1h	WIREFRAME	Any triangle object found to be front-facing is rendered as a series of lines along the triangle boundaries (as determined by the topology type and controlled by the vertex EdgeFlags).	All
2h	POINT	Any triangle object found to be front-facing is rendered as a set of point primitives at the triangle vertices (as determined by the topology type and controlled by the vertex EdgeFlags). NOTE: If the triangle is clipped, points will not be rendered at clip-inserted vertices. Point will only be rendered at original vertices (if visible).	All
3h		Reserved	All

4:3	BackFace Fill Mode Project: All Format: U2 enumerated type FormatDesc This state controls how back-facing triangle and rectangle objects are rendered.

Value	Name	Description	Project
0h	SOLID	Any triangle or rectangle object found to be back-facing is rendered as a solid object. This setting is required when rendering rectangle (RECTLIST) objects.	All
1h	WIREFRAME	Any triangle object found to be back-facing is rendered as a series of lines along the triangle boundaries (as determined by the topology type and controlled by the vertex EdgeFlags).	All
2h	POINT	Any triangle object found to be back-facing is rendered as a set of point primitives at the triangle vertices (as determined by the topology type and controlled by the vertex EdgeFlags). NOTE: If the triangle is clipped, points will not be rendered at clip-inserted vertices. Point will only be rendered at original vertices (if visible).	All
3h		Reserved	All

2	Reserved Project: All Format: MBZ
1	Viewport Transform Enable Project: All Format: Enable FormatDesc This bit controls the Viewport Transform function.
0	Front Winding Project: All Determines whether a triangle object is considered "front facing" if the screen space vertex positions, when traversed in the order, result in a clockwise (CW) or counter-clockwise (CCW) winding order. Does not apply to points or lines.

Value	Name	Description	Project
0h		FRONTWINDING_CW	All
1h		FRONTWINDING_CCW	All

3DSTATE_SF

3 — 31: Anti-aliasing Enable

Project: All

Format: Enable FormatDesc

This field enables “alpha-based” line antialiasing.

Programming Notes
This field must be disabled if any of the render targets have integer (UINT or SINT) surface format.
This field is ignored when **Multisample Rasterization Mode** is MSRASTMODE_ON_xx.

3 — 30:29: Cull Mode

Project: All

Format: 3D_CullMode FormatDesc

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

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

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

3 — 28: Reserved

3 — 27:18: Line Width

Project: All

Format: U3.7 FormatDesc

Range [0.0, 7.9921875]

Controls width of line primitives.

Setting a Line Width of 0.0 specifies the rasterization of the “thinnest” (one-pixel-wide), non-antialiased lines. Note that this effectively overrides the effect of *AAEnable* (though the AAEnable state variable is not modified). Lines rendered with zero Line Width are rasterized using GIQ (Grid Intersection Quantization) rules as specified by the GDI and Direct3D APIs.

Programming Notes	Project
Software must not program a value of 0.0 when running in MSRASTMODE_ON_xxx modes – zero-width lines are not available when multisampling rasterization is enabled.	All

	17:16	Line End Cap Antialiasing Region Width Project: All Format: U2 FormatDesc This field specifies the distances over which the coverage of anti-aliased line end caps are computed. **Note:** this state is duplicated in 3DSTATE_WM. Value | Name | Description | Project 0h | | 0.5 pixels | All 1h | | 1.0 pixels | All 2h | | 2.0 pixels | All 3h | | 4.0 pixels | All
	15:14	Reserved Project: All Format: MBZ
	13	Reserved
	12	Reserved
	11	Scissor Rectangle Enable Project: All Format: Enable FormatDesc Enables operation of Scissor Rectangle.
	10	Reserved Project: All Format: MBZ
	9:8	Multisample Rasterization Mode Project: All Format: U2 enumerated type FormatDesc This state is duplicated in 3DSTATE_WM and both must be set to the same value. See the field in 3DSTATE_WM for definition details.
	7:0	Reserved Project: All Format: MBZ
4	31	Last Pixel Enable Project: All Format: Enable FormatDesc If ENABLED, the last pixel of a diamond line will be lit. This state will only affect the rasterization of Diamond lines (will not affect wide lines or anti-aliased lines). Programming Notes Last pixel is applied to all lines of a LINELIST, and only the last line of a LINESTRIP.

3DSTATE_SF

30:29 Triangle Strip/List Provoking Vertex Select

Project: All

Format: 0-based vertex index FormatDesc

Selects which vertex of a triangle (in a triangle strip or list primitive) is considered the "provoking vertex". Used for flat shading of primitives.

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

28:27 Line Strip/List Provoking Vertex Select

Project: All

Format: 0-based vertex index FormatDesc

Selects which vertex of a line (in a line strip or list primitive) is considered the "provoking vertex".

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

26:25 Triangle Fan Provoking Vertex Select

Project: All

Format: 0-based vertex index FormatDesc

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

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

24:15 Reserved Project: All Format: MBZ

3DSTATE_SF

14 — AA Line Distance Mode

Project: All

Format: U1 FormatDesc

This bit controls the distance computation for antialiased lines.

Value	Name	Description	Project
0h	Reserved		All
1h	AALINEDISTANCE_TRUE	True distance computation. This is the normal setting which should yield WHQL compliance.	All

13 — Reserved Project: All Format: MBZ

12 — Vertex Sub Pixel Precision Select

Project: All

Format: U1 FormatDesc

Selects the number of fractional bits maintained in the vertex data

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

11 — Use Point Width State

Project: All

Format: U1 FormatDesc

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

Value	Name	Description	Project
0h		Use Point Width on Vertex	All
1h		Use Point Width from State	All

10:0 — Point Width

Project: All

Format: U8.3 FormatDesc

Range [0.125, 255.875] pixels

This field specifies the size (width) of point primitives in pixels. This field is overridden (though not overwritten) whenever point width information is passed in the FVF.

3DSTATE_SF		
5	31:0	Global Depth Offset Constant Project: All Format: IEEE_FP FormatDesc Specifies the constant term in the Global Depth Offset function.
6	31:0	Global Depth Offset Scale Project: All Format: IEEE_FP FormatDesc Specifies the scale term used in the Global Depth Offset function.
7	31:0	Global Depth Offset Clamp Project: All Format: IEEE_FP FormatDesc Specifies the clamp term used in the Global Depth Offset function.
8	31	Attribute 1 Component Override W Project: All Format: Enable FormatDesc If set, the W component of output Attribute 1 or 17 (refer to **Number of SF Output Attributes** field for information on which attribute is affected) is overridden by the W component of the constant vector specified by ConstantSource[1].
	30	Attribute 1 Component Override Z Project: All Format: Enable FormatDesc If set, the Z component of output Attribute 1 or 17 (refer to **Number of SF Output Attributes** field for information on which attribute is affected) is overridden by the Z component of the constant vector specified by ConstantSource[1].
	29	Attribute 1 Component Override Y Project: All Format: Enable FormatDesc If set, the Y component of output Attribute 1 or 17 (refer to **Number of SF Output Attributes** field for information on which attribute is affected) is overridden by the Y component of the constant vector specified by ConstantSource[1].
	28	Attribute 1 Component Override X Project: All Format: Enable FormatDesc If set, the X component of output Attribute 1 or 17 (refer to **Number of SF Output Attributes** field for information on which attribute is affected) is overridden by the X component of the constant vector specified by ConstantSource[1].
	27	Reserved Project: All Format: MBZ

26:25 Attribute 1 Constant Source

Project: All

Format: U2 enumerated type FormatDesc

This state selects a constant vector which can be used to override individual components of Attribute 1 or 17 (refer to **Number of SF Output Attributes** field for information on which attribute is affected)

Value	Name	Description	Project
0h	CONST_0000	Constant.xyzw = 0.0,0.0,0.0,0.0	All
1h	CONST_0001_FLOAT	Constant.xyzw = 0.0,0.0,0.0,1.0	All
2h	CONST_1111_FLOAT	Constant.xyzw = 1.0,1.0,1.0,1.0	All
3h	PRIM_ID	Constant.xyzw = PrimID (replicated)	All

24 Reserved Project: All Format: MBZ

23:22 Attribute 1 Swizzle Select

Project: All

Format: U2 enumerated type FormatDesc

This state, along with Attribute 1 Source Attribute, specifies the source for output Attribute 1 or 17 (refer to **Number of SF Output Attributes** field for information on which attribute is affected).

Value	Name	Description	Project
0h	INPUTATTR	This attribute is sourced from AttrInputReg[SourceAttribute]	All
1h	INPUTATTR_FACING	If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute]. If the object is back-facing, this attribute is sourced from AttrInputReg[SourceAttribute+1].	All
2h	INPUTATTR_W	This attribute is sourced from AttrInputReg[SourceAttribute]. The W component is copied to the X component.	All
3h	INPUTATTR_FACING_W	If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute]. If the object is back-facing, this attribute is sourced from AttrInputReg[SourceAttribute+1]. The W component is copied to the X component.	All

3DSTATE_SF		
	21	Reserved Project: All Format: MBZ
	20:16	Attribute 1 Source Attribute Project: All Format: U5 FormatDesc This field selects the source attribute for Attribute 1 or 17 (refer to **Number of SF Output Attributes** field for information on which attribute is affected). Source attribute 0 corresponds to the first 128 bits of data indicated by **Vertex URB Entry Read Offset**
	15	Attribute 0 Component Override W Project: All Format: Enable FormatDesc If set, the W component of output Attribute 0 or 16 (refer to **Number of SF Output Attributes** field for information on which attribute is affected) is overridden by the W component of the constant vector specified by ConstantSource[0].
	14	Attribute 0 Component Override Z Project: All Format: Enable FormatDesc If set, the Z component of output Attribute 0 or 16 (refer to **Number of SF Output Attributes** field for information on which attribute is affected) is overridden by the Z component of the constant vector specified by ConstantSource[0].
	13	Attribute 0 Component Override Y Project: All Format: Enable FormatDesc If set, the Y component of output Attribute 0 or 16 (refer to **Number of SF Output Attributes** field for information on which attribute is affected) is overridden by the Y component of the constant vector specified by ConstantSource[0].
	12	Attribute 0 Component Override X Project: All Format: Enable FormatDesc If set, the X component of output Attribute 0 or 16 (refer to **Number of SF Output Attributes** field for information on which attribute is affected) is overridden by the X component of the constant vector specified by ConstantSource[0].
	11	Reserved Project: All Format: MBZ

10:9 Attribute 0 Constant Source

Project: All

Format: U2 enumerated type FormatDesc

This state selects a constant vector which can be used to override individual components of Attribute 0 or 16 (refer to **Number of SF Output Attributes** field for information on which attribute is affected)

Value	Name	Description	Project
0h	CONST_0000	Constant.xyzw = 0.0,0.0,0.0,0.0	All
1h	CONST_0001_FLOAT	Constant.xyzw = 0.0,0.0,0.0,1.0	All
2h	CONST_1111_FLOAT	Constant.xyzw = 1.0,1.0,1.0,1.0	All
3h	PRIM_ID	Constant.xyzw = PrimID (replicated)	All

8 Reserved Project: All Format: MBZ

7:6 Attribute 0 Swizzle Select

Project: All

Format: U2 enumerated type FormatDesc

This state, along with Attribute 0 Source Attribute, specifies the source for output Attribute 0 or 16 (refer to **Number of SF Output Attributes** field for information on which attribute is affected).

Value	Name	Description	Project
0h	INPUTATTR	This attribute is sourced from AttrInputReg[SourceAttribute]	All
1h	INPUTATTR_FACING	If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute]. If the object is back-facing, this attribute is sourced from AttrInputReg[SourceAttribute+1].	All
2h	INPUTATTR_W	This attribute is sourced from AttrInputReg[SourceAttribute]. The W component is copied to the X component.	All
3h	INPUTATTR_FACING_W	If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute]. If the object is back-facing, this attribute is sourced from AttrInputReg[SourceAttribute+1]. The W component is copied to the X component.	All

3DSTATE_SF		
	5	Reserved Project: All Format: MBZ
	4:0	Attribute 0 Source Attribute Project: All Format: U5 FormatDesc This field selects the source attribute for Attribute 0 or 16 (refer to **Number of SF Output Attributes** field for information on which attribute is affected). Source attribute 0 corresponds to the first 128 bits of data indicated by **Vertex URB Entry Read Offset**
9	31:0	Attribute Control for Attributes 2,3 Project: All Format: see DW 8 FormatDesc
10	31:0	Attribute Control for Attributes 4,5 Project: All Format: see DW 8 FormatDesc
11	31:0	Attribute Control for Attributes 6,7 Project: All Format: see DW 8 FormatDesc
12	31:0	Attribute Control for Attributes 8,9 Project: All Format: see DW 8 FormatDesc
13	31:0	Attribute Control for Attributes 10,11 Project: All Format: see DW 8 FormatDesc
14	31:0	Attribute Control for Attributes 12,13 Project: All Format: see DW 8 FormatDesc
15	31:0	Attribute Control for Attributes 14,15 Project: All Format: see DW 8 FormatDesc

3DSTATE_SF		
16	31:0	Point Sprite Texture Coordinate Enable Project: All Format: 32-bit bitmask FormatDesc When processing point primitives, the attributes from the incoming point vertex are typically copied to the point object corner vertices. However, if a bit is set in this field, the corresponding Attribute is selected as a Point Sprite Texture Coordinate, in which case each corner vertex is assigned a pre-defined texture coordinate as defined by the **Point Sprite Texture Coordinate Origin** state bit. Bit 0 corresponds to output Attribute 0.
17	31:0	Constant Interpolation Enable[31:0] Project: All This field is a bitmask containing a Constant Interpolation Enable bit for each corresponding attribute. If a bit is set, that attribute will undergo constant interpolation, and the corresponding **WrapShortest Enable** bits (if defined) will be ignored. If a bit is clear, components which are not enabled for WrapShortest interpolation (if defined) will be linearly interpolated.
18	31:28	Attribute 7 WrapShortest Enables Project: All Format: 4-bit bitmask FormatDesc This state selects which components (if any) of Attribute 7 or 23 (refer to **Number of SF Output Attributes** field for information on which attribute is affected) are to be interpolated in a “wrap shortest” fashion. Operation is UNDEFINED if any of these bits are set and the **Constant Interpolation Enable** bit associated with this attribute is set. Bit 0: WrapShortest X Component Bit 1: WrapShortest Y Component Bit 2: WrapShortest Z Component Bit 3: WrapShortest W Component
	27:24	Attribute 6 WrapShortest Enables Project: All (See above).
	23:20	Attribute 5 WrapShortest Enables Project: All (See above).
	19:16	Attribute 4 WrapShortest Enables Project: All (See above).
	15:12	Attribute 3 WrapShortest Enables Project: All (See above).

3DSTATE_SF		
	11:8	Attribute 2 WrapShortest Enables Project: All (See above).
	7:4	Attribute 1 WrapShortest Enables Project: All (See above).
	3:0	Attribute 0 WrapShortest Enables Project: All (See above).
19	31:28	Attribute 15 WrapShortest Enables Project: All Format: 4-bit bitmask FormatDesc This state selects which components (if any) of Attribute 15 or 31 (refer to **Number of SF Output Attributes** field for information on which attribute is affected) are to be interpolated in a "wrap shortest" fashion. Operation is UNDEFINED if any of these bits are set and the **Constant Interpolation Enable** bit associated with this attribute is set. Bit 0: WrapShortest X Component Bit 1: WrapShortest Y Component Bit 2: WrapShortest Z Component Bit 3: WrapShortest W Component
	27:24	Attribute 14 WrapShortest Enables Project: All (See above).
	23:20	Attribute 13 WrapShortest Enables Project: All (See above).
	19:16	Attribute 12 WrapShortest Enables Project: All (See above).
	15:12	Attribute 11 WrapShortest Enables Project: All (See above).
	11:8	Attribute 10 WrapShortest Enables Project: All (See above).

3DSTATE_SF		
	7:4	Attribute 9 WrapShortest Enables Project: All (See above).
	3:0	Attribute 8 WrapShortest Enables Project: All (See above).

6.4.2 SF_VIEWPORT [DevSNB]

SF_VIEWPORT			
Project:	DevSNB	**Length Bias:**	2

The viewport-specific state used by the SF unit (SF_VIEWPORT) is stored as an array of up to 16 elements, each of which contains the DWords described below. The start of each element is spaced 8 DWords apart. The location of first element of the array, as specified by **Setup Viewport State Offset**, is aligned to a 32-byte boundary.

[DevSNB+]: This structure contains only the viewport matrix elements (6 dwords). The scissor rectangle parameters are contained in the SCISSOR_RECT state.

DWord	Bit	Description
0	31:0	Viewport Matrix Element m00 Format: Format = IEEE_Float
1	31:0	Viewport Matrix Element m11 Format: Format = IEEE_Float
2	31:0	Viewport Matrix Element m22 Default Value: 0h Excludes DWord (0,1) Format: IEEE_Float
3	31:0	Viewport Matrix Element m30 Format: Format = IEEE_Float
4	31:0	Viewport Matrix Element m31 Format: Format = IEEE_Float
5	31:0	Viewport Matrix Element m32 Format: Format = IEEE_Float

6.4.3 SCISSOR_RECT [DevSNB+]

SCISSOR_RECT		
Project:	[DevSNB+]	
The viewport-specific state used by the SF unit (SCISSOR_RECT) is stored as an array of up to 16 elements, each of which contains the DWords described below. The start of each element is spaced 2 DWords apart. The location of first element of the array, as specified by **Pointer to SCISSOR_RECT**, is aligned to a 32-byte boundary.		
DWord	**Bit**	**Description**
0	31:16	Scissor Rectangle Y Min Project: All Format: U16 Pixels from Drawing Rectangle origin (upper left corner) Range [DevSNB]: 0..8191 Specifies Y Min coordinate of (inclusive) Scissor Rectangle used for scissor test. Pixels with (Draw Rectangle-relative) Y coordinates *less than* **Y Min** will be clipped out if Scissor Rectangle is enabled. NOTE: If Y Min is set to a value greater than Y Max, all primitives will be discarded for this viewport.
	15:0	Scissor Rectangle X Min Project: All Format: U16 Pixels from Drawing Rectangle origin (upper left corner) Range [DevSNB]: 0..8191 Specifies X Min coordinate of (inclusive) Scissor Rectangle used for scissor test. Pixels with (Draw Rectangle-relative) X coordinates *less than* **X Min** will be clipped out if Scissor Rectangle is enabled. NOTE: If X Min is set to a value greater than X Max, all primitives will be discarded for this viewport.
1	31:16	Scissor Rectangle Y Max Project: All Format: U16 Pixels from Drawing Rectangle origin (upper left corner) Range [DevSNB]: 0..8191 Specifies Y Max coordinate of (inclusive) Scissor Rectangle used for scissor test. Pixels with (Draw Rectangle-relative) Y coordinates *greater than* **Y Max** will be clipped out if Scissor Rectangle is enabled.

SCISSOR_RECT		
	15:0	Scissor Rectangle X Max Project: All Format: U16 Pixels from Drawing Rectangle origin (upper left corner) Range 0[DevSNB]: 0..8191 Specifies X Max coordinate of (inclusive) Scissor Rectangle used for scissor test. Pixels with (Draw Rectangle-relative) Y coordinates *greater than* **X Max** will be clipped out if Scissor Rectangle is enabled.

6.5 Attribute Interpolation Setup [DevSNB+]

With the attribute interpolation setup function being implemented in hardware for [DevSNB+], a number of state fields in 3DSTATE_SF are utilized to control interpolation setup.

Number of SF Output Attributes sets the number of attributes that will be output from the SF stage, not including position. This can be used to specify up to 32, and may differ from the number of input attributes. The number of input attributes is derived from the **Vertex URB Entry Read Length** field. Note that this field is also used to specify whether swizzling is to be performed on Attributes 0-15 or Attributes 16-32. See the state field definition for details.

6.5.1 Attribute Swizzling

The first or last set of 16 attributes can be swizzled according to certain state fields. **Attribute Swizzle Enable** enables the swizzling for all 16 of these attributes, and each of the attributes has a 2-bit **Swizzle Select** field that controls swizzling with the following settings:

- INPUTATTR – This attribute is sourced from AttrInputReg[SourceAttribute].
- INPUTATTR_FACING – This attribute is sourced from AttrInputReg[SourceAttribute] if the object is front-facing, otherwise it is sourced from AttrInputReg[SourceAttribute+1].
- INPUTATTR_W – This attribute is sourced from AttrInputReg[SourceAttribute]. WYZW (the W component of the source is copied to the X component of the destination).
- INPUTATTR_FACING – If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute]. WYZW (the W component of the source is copied to the X component of the destination). If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute+1]. WYZW.

Each of the first or last set of 16 attributes also has a 5-bit **Source Attribute** field which specify, per output attribute (not component), which input attribute sources the output attribute when INPUTATTR is selected for **Swizzle Select**. A **Source Attribute** value of 0 corresponds to the 128-bit attribute immediately following the vertex 4D position. If INPUTATTR_FACING is selected, this specifies the first of two consecutive (front,back) input attributes, where the SourceAttribute value can be an odd or even number (just not 31, as that would place the back-face input attribute past the end of the input max complement of input attributes).

Constant overriding is also available for the first or last set of 16 attributes. Each attribute has a **Constant Source** field which specifies the constant values per swizzled attribute, with the following settings available:

- XYZW = 0000
- XYZW = 0001
- XYZW = 1111

Each channel of each attribute has a **Component Override** field to control whether the corresponding channel is overridden with the constant value defined in **Constant Source**.

6.5.2 Interpolation Modes

All 32 attributes have a **Constant Interpolation Enable** state field bit to specify whether all components of the post-swizzled attribute are to be interpolated as constant values (not varying over the pixels of the object). If set, the attribute at the provoking vertex is copied to a0, and a1 and a2 are set to zero – this results in a constant interpolation of the provoking vertex value. If clear, the attribute is linearly interpolated. Attributes 0-15 are further subjected to Wrap Shortest processing on a per-component basis, via the **Attribute WrapShortest Enables** state bitfields. WrapShortest processing modifies the a1 and/or a2 values depending on attribute deltas.

The table below indicates the output values of a0, a1, and a2 depending on interpolation mode settings.

	a0	**a1**	**a2**
Constant	A0	0.0	0.0
Linear	A0	A1-A0	A2-A0
Wrap Shortest	A0	(A1-A0)+1 (A1-A0) <= -0.5 (A1-A0)-1 (A1-A0) >= 0.5 (A1-A0) otherwise	(A2-A0)+1 (A2-A0) <= -0.5 (A2-A0)-1 (A2-A0) >= 0.5 (A2-A0) otherwise

6.5.3 Point Sprites

Normally all vertex attributes (including texture coordinates) other than position are simply replicated from the incoming point center vertex to the generated point object (corner) vertices. However, OGL supports "sprite points", where some/all texture coordinates are replaced with full-scale 2D texture coordinates.

A 32-bit **PointSprite TextureCoordinate Enable** bit mask controls whether the corresponding vertex attribute is to be replaced by a sprite point texture coordinate. The global (not per-attribute) **Point Sprite TextureCoordinate Origin** field controls how the point object vertex (top/bottom, left/right) texture coordinates are generated:

UPPERLEFT	Left	Right
Top	(0,0,0,1)	(1,0,0,1)

Bottom	(0,1,0,1)	(1,1,0,1)

LOWERLEFT	Left	Right
Top	(0,1,0,1)	(1,1,0,1)
Bottom	(0,0,0,1)	(1,0,0,1)

6.6 Depth Offset [DevSNB+]

The state for depth offset in 3DSTATE_SF controls the depth offset function. Since this function was previously contained in the Windower stage, refer to the "Depth Offset" section in the Windower chapter for more details on this function.

6.7 Other SF Functions

6.7.1 Statistics Gathering

The SF stage itself does not have any associated pipeline statistics; however, it counts the number of objects being output by the clipper on the clipper's behalf, since it less feasible to have the CLIP unit figure out how many objects have been output by a clip thread. It is easy for the SF unit to count the number of objects it receives from the CLIP stage since it is decomposing the output primitive topologies into objects anyway.

If the **Statistics Enable** bit is set in SF_STATE, then SF will increment the CL_PRIMITIVES_COUNT Register (see Memory Interface Registers in Volume Ia, *GPU*) once for each object in each primitive topology it receives from the CLIP stage. This bit should always be set if clipping is enabled and pipeline statistics are desired.

Software should always clear the **Statistics Enable** bit in SF_STATE if the clipper is disabled since objects SF receives are not considered "primitives output by the clipper" unless the clipper is enabled. Note that the clipper can be disabled either using bypass mode via a PIPELINE_STATE_POINTERS command with **Clip Enable** clear *or* by setting **Clip Mode** in CLIP_STATE to CLIPMODE_ACCEPT_ALL.

7. Windower (WM) Stage

7.1 Overview

As mentioned in the *SF Unit* chapter, the SF stage prepares an object for scan conversion by the Window/Masker (WM) unit. Refer to the *SF Unit* chapter for details on the screen-space geometry of objects to be rendered. The WM unit uses the parameters provided by the SF unit in the object-specific rasterization algorithms.

The WM stage of the 3D pipeline performs the following operations (at a high level)

- Pre-scan-conversion modification of some primitive attributes, including
  - Application of Depth Offset to the position Z attribute
- Scan-conversion of the various primitive types, including
  - 2D clipping to the scissor/draw rectangle intersection
- Spawning of Pixel Shader (PS) threads to process the pixels resulting from scan-conversion

The spawned Pixel Shader (PS) threads are responsible for the following (high-level) operations

- interpolation of vertex attributes (other than X,Y,Z) to the pixel location
- performing any "Pixel Shader" operations dictated by the API PS program
  - Using the Sampler shared function to sample data from "texture" surfaces
  - Using the DataPort to perform general memory I/O
- Submitting the shaded pixel results to the DataPort for any subsequent "blending" (aka Output Merger) operation and write to the RenderCache.

The WM unit keeps a scoreboard of pixels being processed in outstanding PS threads in order to guarantee in-order rasterization results. This allows the WM unit to overlap processing of several objects.

7.1.1 Inputs from SF to WM

The outputs from the SF stage to the WM stage are mostly comprised of implementation-specific information required for the rasterization of objects. The types of information is summarized below, but as the interface is not exposed to software a detailed discussion is not relevant to this specification.

- PrimType of the object
- VPIndex, RTAIndex associated with the object
- Handle of the Primitive URB Entry (PUE) that was written by the SF (Setup) thread. This handle will be passed to all WM (PS) threads spawned from the WM's rasterization process.
- Information regarding the X,Y extent of the object (e.g., bounding box, etc.)
- Edge or line interpolation information (e.g., edge equation coefficients, etc.)

- Information on where the WM is to start rasterization of the object
- Object orientation (front/back-facing)
- Last Pixel indication (for line drawing)

7.2 Windower Pipelined State

7.2.1 3DSTATE_WM

7.2.1.1 3DSTATE_WM [DevSNB]

For **[DevSNB]**, the state used by the windower stage is defined with this inline state packet.

3DSTATE_WM			
Project:	[DevSNB]	**Length Bias:**	2

DWord	Bit	Description
0	31:29	Command Type Default Value: 3h GFXPIPE Format: OpCode
	28:27	Command SubType Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	3D Command Opcode Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	3D Command Sub Opcode Default Value: 14h 3DSTATE_WM Format: OpCode
	15:8	Reserved Project: All Format: MBZ
	7:0	DWord Length Default Value: 07h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:6	Kernel Start Pointer[0] Project: All Address: InstructionBaseOffset[31:6] Surface Type: Kernel Specifies the 64-byte aligned address offset of the first instruction in the kernel[0]. This pointer is relative to the **Instruction Base Address**.
	5:0	Reserved Project: All Format: MBZ

3DSTATE_WM

2	31	Single Program Flow (SPF) Project: All Specifies the initial condition of the kernel program as either a single program flow (SIMDnxm with m = 1) or as multiple program flows (SIMDnxm with m > 1). See CR0 description in *ISA Execution Environment*.

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

	30	Vector Mask Enable (VME) Project: All Format: U1 enumerated type FormatDesc When SPF=0, VME specifies which mask to use to initialize the initial channel enables. When SPF=1, VME specifies which mask to use to generate execution channel enables.

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

	29:27	Sampler Count Project: All Format: U3 FormatDesc Range [0,4] Specifies how many samplers (in multiples of 4) the vertex shader 0 kernel uses. Used only for prefetching the associated sampler state entries.

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

	26	Reserved Project: All Format: MBZ

3DSTATE_WM

25:18 — Binding Table Entry Count

Project: All

Format: U8 FormatDesc

Range [0,255]

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

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

See *3D Pipeline* for more information.

17 — Thread Priority

Project: All

Specifies the priority of the thread for dispatch.

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

16 — Floating Point Mode

Project: All

Specifies the floating point mode used by the dispatched thread.

Value	Name	Description	Project
0h	IEEE-745	Use IEEE-754 rules	All
1h	Alt	Use alternate rules	All

15:14 — Reserved Project: All Format: MBZ

13 — Illegal Opcode Exception Enable

Project: All

Format: Enable FormatDesc

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

12 — Reserved Project: All Format: MBZ

11 — MaskStack Exception Enable

Project: All

Format: Enable FormatDesc

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

10:8 — Reserved Project: All Format: MBZ

3DSTATE_WM		
	7	Software Exception Enable Project: All Format: Enable FormatDesc This bit gets loaded into EU CR0.1[13] (note the bit # difference). See *Exceptions* and *ISA Execution Environment*.
	6:0	Reserved Project: All Format: MBZ
3	31:10	Scratch Space Base Pointer Project: All Address: GeneralStateOffset[31:10] Surface Type: ScratchSpace Specifies the 1k-byte aligned address offset to scratch space for use by the kernel. This pointer is relative to the **General State Base Address**.
	9:4	Reserved Project: All Format: MBZ
	3:0	Per Thread Scratch Space Project: All Format: U4 FormatDesc Range [0,11] indicating [1k bytes, 2M bytes] in powers of two Specifies the amount of scratch space allowed to be used by each thread. The driver must allocate enough contiguous scratch space, pointed to by the **Scratch Space Pointer**, to ensure that the **Maximum Number of Threads** each get **Per Thread Scratch Space** size without exceeding the driver-allocated scratch space.

3DSTATE_WM

4	31	Statistics Enable Project: All Format: Enable FormatDesc If ENABLED, the Windower and pixel pipeline will engage in statistics gathering. If DISABLED, statistics information associated with this FF stage will be left unchanged. See *Statistics Gathering*.

Programming Notes	Project
This bit should be *set* for “normal” operation since statistics are supposed to be continuously calculated. This bit must be disabled if either of these bits is set: Depth Buffer Clear , Hierarchical Depth Buffer Resolve Enable or Depth Buffer Resolve Enable. DevSNBCO+ Errata: Software should also set the mmio register 0x2084[7] to “1”, in cases where the "depth statistics bit" is DISABLED, HIZ buffer is ENABLED and NONE of these bits are set (**Depth Buffer Clear , Hierarchical Depth Buffer Resolve Enable or Depth Buffer Resolve Enable)** **Note**:- The mmio register 0x2084[7] must only be programmed when the 3D pipe is IDLE **Note:-** The mmio register 0x2084[7] must only be programmed when the 3D pipe is IDLE	All

	30	Depth Buffer Clear Project: All Format: Enable FormatDesc When set, the depth buffer is initialized as a side-effect of rendering pixels.

Programming Notes	Project
If this field is enabled, the **Depth Test Enable** field in DEPTH_STENCIL_STATE must be disabled	All
Refer to section 0 “Depth Buffer Clear” for additional restrictions when this field is enabled.	All

	29	Reserved Project: All Format: MBZ

3DSTATE_WM

	28	Depth Buffer Resolve Enable Project: All Format: Enable FormatDesc When set, the depth buffer is made to be consistent with the hierarchical depth buffer as a side-effect of rendering pixels. This is intended to be used when the depth buffer is to be used as a surface outside of the 3D rendering operation. **Programming Notes** — **Project** If this field is enabled, the **Depth Buffer Clear** and **Hierarchical Depth Buffer Resolve Enable** fields must both be disabled. Refer to section 7.5.3.2 "Depth Buffer Resolve" for additional restrictions when this field is enabled. — All If **Hierarchical Depth Buffer Enable** is disabled, enabling this field will have no effect. — All
	27	Hierarchical Depth Buffer Resolve Enable Project: All Format: Enable FormatDesc When set, the hierarchical depth buffer is made to be consistent with the depth buffer as a side-effect of rendering pixels. This is intended to be used when the depth buffer has been modified outside of the 3D rendering operation. **Programming Notes** — **Project** If this field is enabled, the **Depth Buffer Clear** and **Depth Buffer Resolve Enable** fields must both be disabled. Refer to section 7.5.3.3 "Hierarchical Depth Buffer Resolve" for additional restrictions when this field is enabled. — All If **Hierarchical Depth Buffer Enable** is disabled, enabling this field will have no effect. — All **Performance Note:** expect the hierarchical depth buffer's impact on performance to be reduced for some period of time after this operation is performed, as the hierarchical depth buffer is initialized to a state that makes it ineffective. Further rendering will tend to bring the hierarchical depth buffer back to a more effective state. — All
	26:23	Reserved Project: All Format: MBZ
	22:16	Dispatch GRF Start Register for Constant/Setup Data [0] Project: All Format: U7 FormatDesc Range [0,127] Specifies the starting GRF register number for the Constant/Setup portion of the thread payload for kernel[0].
	15	Reserved Project: All Format: MBZ

3DSTATE_WM		
	14:8	Dispatch GRF Start Register for Constant/Setup Data [1] Project: All Format: U7 FormatDesc Range [0,127] Specifies the starting GRF register number for the Constant/Setup portion of the thread payload for kernel[1].
	7	Reserved Project: All Format: MBZ
	6:0	Dispatch GRF Start Register for Constant/Setup Data [2] Project: All Format: U7 FormatDesc Range [0,127] Specifies the starting GRF register number for the Constant/Setup portion of the thread payload for kernel[2].
5	31:25	Maximum Number of Threads Project: All Format: U7 representing (thread count – 1) FormatDesc Range: [DevSNB:B0] with **WIZ Hashing Disable** in GT_MODE register *enabled*: Range = [3,79] □ [4,80] threads. Only odd values are allowed (resulting in even max number of threads) [DevSNB:A0] with **WIZ Hashing Disable** in GT_MODE register *enabled*: Range = [5,79] □ [6,80] threads. Only odd values are allowed (resulting in even max number of threads) [DevSNB] with **WIZ Hashing Disable** in GT_MODE register *disabled*: Range = [1,39] □ [2,40] threads **Programming Notes** A PIPE_CONTROL command, with only the Stall At Pixel Scoreboard field set (DW1 Bit 1), must be issued prior to any change to the value in this field
	24	Reserved Project: All Format: MBZ
	23	Legacy Diamond Line Rasterization Project: All Format: Enable FormatDesc This bit, if ENABLED, indicates that the Windower will rasterize zero width lines using the DX9 rasterization rules. If DISABLED, the Windower will rasterize zero width lines using the DX10 rasterization rules (see *Strips Fans* chapter).

	22	Pixel Shader Kill Pixel Project: All Format: Enable FormatDesc This bit, if ENABLED, indicates that the PS kernel or color calculator has the ability to kill (discard) pixels or samples, *other than due to depth or stencil testing*. This bit is required to be ENABLED in the following situations: The API pixel shader program contains “killpix” or “discard” instructions, or other code in the pixel shader kernel that can cause the final pixel mask to differ from the pixel mask received on dispatch. A sampler with chroma key enabled with kill pixel mode is used by the pixel shader. Any render target has Alpha Test Enable or AlphaToCoverage Enable enabled. The pixel shader kernel generates and outputs oMask. Note: As ClipDistance clipping is fully supported in hardware and therefore not via PS instructions, there should be no need to ENABLE this bit due to ClipDistance clipping.
	21	Pixel Shader Computed Depth Project: All Format: Enable FormatDesc This bit, if ENABLED, indicates that the PS kernel computes and outputs a depth value. Programming Notes — Project If a NULL Depth Buffer is selected, the **Pixel Shader Computed Depth** field must be set to disabled. — All
	20	Pixel Shader Uses Source Depth Project: All Format: Enable FormatDesc This bit, if ENABLED, indicates that the PS kernel requires the source depth value (vPos.z) to be passed in the payload. The source depth value is interpolated according to the **Position ZW Interpolation Mode** state. [DevSNB-A/B] Errata: This bit must be reset when Depth Bufer Surface Format is D32_FLOAT_S8X24_UINT and Number of Multisamples = 4.

19	Thread Dispatch Enable Project: All Format: Enable FormatDesc This bit, if set, indicates that it is possible for a PS thread to modify a render target, i.e.,at least one render target is enabled (is not of type SURFTYPE_NULL and has at least one channel enabled for writes) and the PS kernel contains a code path that may issue a write to that/those enabled RTs.

Programming Notes	Project
This bit is used for performance optimizations and does not directly control writing to render targets. If this bit is DISABLED, no pixel shader threads will be dispatched.	All
For correct behavior, this bit must be set consistently with the behavior of the PS kernel, i.e. if this bit is DISABLED the PS kernel must not write color or depth to any render targets.	All
If this field is disabled, **Pixel Shader Kill Pixel** must be disabled.	

18	Reserved Project: All Format: MBZ
17:16	Line End Cap Antialiasing Region Width Project: All Format: U2 FormatDesc This field specifies the distances over which the coverage of anti-aliased line end caps are computed. **Note:** This state is duplicated in 3DSTATE_SF.

Value	Name	Description	Project
0h		0.5 pixels	All
1h		1.0 pixels	All
2h		2.0 pixels	All
3h		4.0 pixels	All

15:14	Line Antialiasing Region Width Project: All Format: U2 FormatDesc This field specifies the distance over which the anti-aliased line coverage is computed.

Value	Name	Description	Project
0h		0.5 pixels	All
1h		1.0 pixels	All
2h		2.0 pixels	All
3h		4.0 pixels	All

3DSTATE_WM		
	13	Polygon Stipple Enable Project: All Format: Enable FormatDesc Enables the Polygon Stipple function.
	12	Reserved Project: All Format: MBZ
	11	Line Stipple Enable Project: All Format: Enable FormatDesc Enables the Line Stipple function.
	10	Reserved Project: All Format: MBZ
	9	oMask Present to RenderTarget Project: All Format: Enable FormatDesc This bit is inserted in the PS payload header and made available to the DataPort (either via the message header or via header bypass) to indicate that oMask data (one or two phases) is included in Render Target Write messages. If present, the oMask data is used to mask off samples. [DevSNB:A0] This bit must be disabled in A Step.
	8	Pixel Shader Uses Source W Project: All Format: Enable FormatDesc This bit, if ENABLED, indicates that the PS kernel requires the interpolated source W value (vPos.w) to be passed in the payload. The W value is interpolated according to the **Position ZW Interpolation Mode** state.
	7	Dual Source Blend Enable Project: All Format: Enable FormatDesc This field is set if dual source blend is enabled. If this bit is disabled, the data port dual source message reverts to a single source message using source 0. [DevSNB:A0] : Errata : This bit must be set to 0. Dual Source Blending is not supported in DevSNB A Step.
	6:3	Reserved Project: All Format: MBZ

3DSTATE_WM		
	2	32 Pixel Dispatch Enable Project: All Format: Enable FormatDesc Enables the Windower to dispatch 8 subspans in one payload. Note: in the table below, the Valid column indicates which products that combination is supported on. Combinations of dispatch enables not listed in the table are not available on any product. A: Valid on all products B: Not valid on [DevSNB] if 4x PERPIXEL mode with pixel shader computed depth. D: Valid on all products, except when in non-1x PERSAMPLE mode (applies to [DevSNB+] only). Not valid on [DevSNB] if 4x PERPIXEL mode with pixel shader computed depth. E: Not valid on [DevSNB] if 4x PERPIXEL mode with pixel shader computed depth. F: Valid on all products, except not valid on [DevSNB] if 4x PERPIXEL mode with pixel shader computed depth. For **[DevSNB],** there is only one kernel start pointer (KSP) specified in WM_STATE, with other kernels being entered via an offset **from the single KSP as follows:** SP[0] = **KSP** KSP[1] = KSP+1 KSP[2] = KSP+2 KSP[3] = KSP+3 For **[DevSNB]**, each of the three KSP values is separately specified. In addition, each kernel has a separately-specified GRF register count. Table 22 for valid pixel dispatch combinations.

3DSTATE_WM		
	1	16 Pixel Dispatch Enable Project: All Format: Enable FormatDesc Enables the Windower to dispatch 4 subspans in one payload. Note: in the table below, the Valid column indicates which products that combination is supported on. Combinations of dispatch enables not listed in the table are not available on any product. A: Valid on all products B: Not valid on [DevSNB] if 4x PERPIXEL mode with pixel shader computed depth. D: Valid on all products, except when in non-1x PERSAMPLE mode (applies to [DevSNB+] only). Not valid on [DevSNB] if 4x PERPIXEL mode with pixel shader computed depth. E: Not valid on [DevSNB] if 4x PERPIXEL mode with pixel shader computed depth. F: Valid on all products, except not valid on [DevSNB] if 4x PERPIXEL mode with pixel shader computed depth. For [DevSNB], there is only one kernel start pointer (KSP) specified in WM_STATE, with other kernels being entered via an offset from the single KSP as follows: SP[0] = KSP KSP[1] = KSP+1 KSP[2] = KSP+2 KSP[3] = KSP+3 For [DevSNB], each of the three KSP values is **separately specified. In addition, each kernel has a separately-specified GRF register count.** Table 22 for valid pixel dispatch combinations.

3DSTATE_WM		
	0	8 Pixel Dispatch Enable Project: All Format: Enable FormatDesc Enables the Windower to dispatch 2 subspans in one payload. **Note:** See Note: in the table below, the Valid column indicates which products that combination is supported on. Combinations of dispatch enables not listed in the table are not available on any product. **A:** Valid on all products **B:** Not valid on **[DevSNB]** if 4x PERPIXEL **mode** with pixel shader computed depth. D: Valid on all **products, except when in non-1x PERSAMPLE mode (applies to [DevSNB+] only). Not valid on [DevSNB] if 4x PERPIXEL mode with pixel shader computed depth.** **E:** Not valid on [DevSNB] if 4x PERPIXEL *mode* with pixel shader computed **depth.** F: Valid on all products, **except** not valid on [DevSNB] if 4x PERPIXEL mode **with pixel shader computed depth.** **For** [DevSNB], there is **only** one kernel start pointer (KSP) specified in WM_STATE, with **other kernels being entered via an offset from the single KSP as follows:** **SP[0]** = KSP KSP[1] = KSP+1 KSP[2] = KSP+2 KSP[3] = **KSP+3** For [DevSNB], each of the three KSP values is separately specified. In addition, each kernel has a separately-specified GRF register count. Table **22** for valid pixel dispatch combinations.
6	31:26	Reserved Project: All Format: MBZ
	25:20	Number of SF Output Attributes Project: All Format: U6 count of attributes FormatDesc Range [0,32] Specifies the number of vertex attributes passed from the SF stage to the WM stage (does not include Position). This field must reflect the same number of actual attributes as the **Number of SF Output Attributes** field in 3DSTATE_SF.

3DSTATE_WM

	19:18	Position XY Offset Select Project: All Format: U2 enumerated type FormatDesc This field specifies if/what **Position XY Offset** values are passed in the PS payload. Note that these are per-slot (pixel|sample) offsets, and therefore separate from the subspan XY coordinates passed in R1.

Value	Name	Description	Project
0h	POSOFFSET _NONE	No Position XY Offsets are included in the PS payload.	All
1h	Reserved		All
2h	POSOFFSET _CENTROID	Position XY Offsets will be passed in the PS payload, and these will reflect the Centroid position(s).	All
3h	POSOFFSET _SAMPLE	Position XY Offsets will be passed in the PS payload, and these will reflect the multisample position(s).	All

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

3DSTATE_WM

	17:16	Position ZW Interpolation Mode Project: All Format: U2 enumerated type FormatDesc This field elects “interpolation mode” associated with the Position Z (source depth) and W coordinates passed in the PS payload when the PS requires Position as input. This field does not determine whether these coordinates are actually included in the payload (see **Pixel Shader Requires Depth**, **Pixel Shader Requires W**).

Value	Name	Description	Project
0h	INTERP_PIXEL	Evaluate Z & W at the pixel center or UL corner (as specified by **Pixel Location** of 3DSTATE_MULTISAMPLE)	All
1h	Reserved		All
2h	INTERP_CENTROID		All
3h	INTERP_SAMPLE		All

Programming Notes	Project
Position XY Offset Select controls if/what Position XY Offset values are included in the PS payload. This field (**Position ZW Interpolation Mode**) does not affect those Position XY Offset values.	All
The device may need to evaluate Z and W coordinates using different interpolation locations, for DepthTest and barycentric coordinate generation. This field only impacts the values that get passed in the PS payload.	All
MSDISPMODE_PERSAMPLE is required in order to select INTERP_SAMPLE.	All
[DevSNB-A,B{W/A}]: Erratum: For 4X MSRT, when Pixel Shader Uses Source Depth is enabled, this bit-field must not be INTERP_PIXEL.	[DevSNB-A,B]

3DSTATE_WM

15:10 Barycentric Interpolation Mode

Project: All

Format: 6-bit enable mask FormatDesc

Controls which barycentric interpolation terms must be passed into the pixel shader kernel.

Bit 0: Perspective Pixel Location barycentric is required

Bit 1: Perspective Centroid barycentric is required

Bit 2: Perspective Sample barycentric is required

Bit 3: Non-perspective Pixel Location barycentric is required

Bit 4: Non-perspective Centroid barycentric is required

Bit 5: Non-perspective Sample barycentric is required

[DevSNB:A0{WKA}]: [DevSNB:A]: Errata: Perspective Pixel Location Barycentric (Bit 0) and Non-perspective Pixel Location Barycentric (Bit 3) are not supported with Number of Multisamples equals to 4 in 3DSTATE_MULTISAMPLE

Programming Notes	Project
Pixel Location below refers to either the upper left corner or pixel center depending on the **Pixel Location** state of 3DSTATE_MULTISAMPLING).	All
MSDISPMODE_PERSAMPLE is required in order to select Perspective Sample or Non-perspective Sample barycentric coordinates.	All

9 Point Rasterization Rule

Project: All

Format: 3D_RasterizationRule FormatDesc

This field specifies the rasterization rules to be applied whenever the edges of a point primitive fall exactly on a pixel sampling point.

Value	Name	Description	Project
0h	RASTRULE_UPPER_LEFT	To match “normal” upper left rules for surface primitives	All
1h	RASTRULE_UPPER_RIGHT	To match OpenGL point rasterization rules (round to + infinity, where this is the upper right direction wrt OpenGL screen origin of lower left).	All

8:3 Reserved Project: All Format: MBZ

2:1 Multisample Rasterization Mode

Project: All

Format: U2 enumerated type FormatDesc

This field determines whether multisample rasterization is turned on/off, and how the pixel sample point(s) are defined. Software sets this according to the API, the API's multisample enable state setting (if any), and whether 1X or 4X MSRTs are bound. This state is duplicated in 3DSTATE_SF and both must be set to the same value. Refer to the "Multisampling" section for details on the settings of this field.

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

Programming Notes
Setting this field to MSRASTMODE_xxx_PATTERN when **Number of Multisamples** == NUMSAMPLES_1 is UNDEFINED.

0 Multisample Dispatch Mode

Project: All

Format: U1 enumerated type FormatDesc

This bit, along with **Number of Multisamples**, determines how PS threads are dispatched. Software programs this bit depending on the per-pixel v.s per-sample PS execution requirement.

Value	Name	Description	Project
0h	MSDISPMODE _PERSAMPLE	This is the high-quality multisample mode where (over and above PERPIXEL mode) the PS is run for each covered sample. This mode is also used for "normal" non-multisample rendering (aka 1X), given **Number of Multisamples** is programmed to NUMSAMPLES_1.	All
1h	MSDISPMODE _PERPIXEL	This is the classic multisample mode of operation, typically used for both antialiasing and transparency. Setup and rasterization operate in full multisample mode, testing coverage and depth/stencil test at the sample level but only running the PS once per pixel.	All

3DSTATE_WM		
7	31:6	Kernel Start Pointer[1] Project: All Address: InstructionBaseOffset[31:6] Surface Type: Kernel Specifies the 64-byte aligned address offset of the first instruction in kernel[1]. This pointer is relative to the **Instruction Base Address**.
	5:0	Reserved Project: All Format: MBZ
8	31:6	Kernel Start Pointer[2] Project: All Address: InstructionBaseOffset[31:6] Surface Type: Kernel Specifies the 64-byte aligned address offset of the first instruction in kernel[2]. This pointer is relative to the **Instruction Base Address**.
	5:0	Reserved Project: All Format: MBZ

7.2.2 3DSTATE_CONSTANT_PS [DevSNB]

3DSTATE_CONSTANT_PS			
Project:	[DevSNB]	**Length Bias:**	2

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

[DevSNB:A] All memory accesses are to GGTT address space, independent of the PPGTT mode bit in GFX_MODE

[DevSNB]: This packet must be followed by WM_STATE.

Programming Note:

- It is invalid to program this command mroe then once between 3D_PRIMITIVE commands.

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

3DSTATE_CONSTANT_PS		
	23:16	3D Command Sub Opcode Default Value: 17h 3DSTATE_CONSTANT_PS Format: OpCode
	15	Buffer 3 Valid Project: [DevSNB] Format: Enable This field enables buffer 3
	14	Buffer 2 Valid Project: [DevSNB] Format: Enable This field enables buffer 2
	13	Buffer 1 Valid Project: [DevSNB] Format: Enable This field enables buffer 1
	12	Buffer 0 Valid Project: [DevSNB] Format: Enable This field enables buffer 0
	11:8	Constant Buffer Object Control State Project: [DevSNB] Format: MEMORY_OBJECT_CONTROL_STATE FormatDesc Specifies the memory object control state for all constant buffers defined in this command. [DevSNB-A] Only bits 11 and 10 can be used (GFDT)
	7:0	DWord Length Default Value: 3h Excludes DWord (0,1) Format: =n Total Length - 2 Project: [DevSNB]
1	31:5	Pointer to PS Constant Buffer 0 Project: [DevSNB] Address: DynamicStateOffset[31:5] or GraphicsAddress[31:5] Surface Type: ConstantBuffer This field points to the location of PS Constant Buffer 0. The state of **INSTPM<CONSTANT_BUFFER Address Offset Disable>** determines whether the Dynamic State Base Address is added to this pointer. Programming Notes Constant buffers must be allocated in linear (not tiled) graphics memory.

3DSTATE_CONSTANT_PS		
	4:0	PS Constant Buffer 0 Read Length Project: [DevSNB] Format: U5 (read length – 1) FormatDesc This field specifies the length of the constant data to be loaded from memory in 256-bit units minus one. Programming Notes The sum of all four read length fields (each incremented to represent the actual read length) must be less than or equal to 64
2	31:5	Pointer to PS Constant Buffer 1 Project: [DevSNB] Address: GraphicsAddress[31:5] Surface Type: ConstantBuffer This field points to the location of PS Constant Buffer 1. Programming Notes Constant buffers must be allocated in linear (not tiled) graphics memory.
	4:0	PS Constant Buffer 1 Read Length Project: [DevSNB] Format: U5 (read length – 1) FormatDesc This field specifies the length of the constant data to be loaded from memory in 256-bit units minus one. Programming Notes The sum of all four read length fields (each incremented to represent the actual read length) must be less than or equal to 64
3	31:5	Pointer to PS Constant Buffer 2 Project: [DevSNB] Address: GraphicsAddress[31:5] Surface Type: ConstantBuffer This field points to the location of PS Constant Buffer 2. Programming Notes Constant buffers must be allocated in linear (not tiled) graphics memory.

3DSTATE_CONSTANT_PS		
	4:0	PS Constant Buffer 2 Read Length Project: [DevSNB] Format: U5 (read length – 1) FormatDesc This field specifies the length of the constant data to be loaded from memory in 256-bit units minus one. Programming Notes The sum of all four read length fields (each incremented to represent the actual read length) must be less than or equal to 64
4	31:5	Pointer to PS Constant Buffer 3 Project: [DevSNB] Address: GraphicsAddress[31:5] Surface Type: ConstantBuffer This field points to the location of PS Constant Buffer 3. Programming Notes Constant buffers must be allocated in linear (not tiled) graphics memory.
	4:0	PS Constant Buffer 3 Read Length Project: [DevSNB] Format: U5 (read length – 1) FormatDesc This field specifies the length of the constant data to be loaded from memory in 256-bit units minus one. Programming Notes The sum of all four read length fields (each incremented to represent the actual read length) must be less than or equal to 64

7.2.3 3DSTATE_SAMPLE_MASK [DevSNB+]

For **[DevSNB+]**, the sample mask state used by the windower stage is defined with this inline state packet.

3DSTATE_SAMPLE_MASK			
Project:	[DevSNB+]	**Length Bias:**	2

DWord	Bit	Description
0	31:29	Command Type Default Value: 3h GFXPIPE Format: OpCode
	28:27	Command SubType Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	3D Command Opcode Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	3D Command Sub Opcode Default Value: 18h 3DSTATE_SAMPLE_MASK Format: OpCode
	15:8	Reserved Project: All Format: MBZ
	7:0	DWord Length Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:4	Reserved Project: DevSNB Format: MBZ
	3:0	Sample Mask Project: DevSNB Format: Right-justified bitmask (Bit 0 = Sample0). Number of bits that are used is determined by **Num Multisamples** (3DSTATE_MULTISAMPLE) FormatDesc A per-multisample-position mask state variable that is immediately and unconditionally ANDed with the sample coverage mask as part of the rasterization process. This mask is applied prior to centroid selection.
	31:8	Reserved Project: Format: MBZ

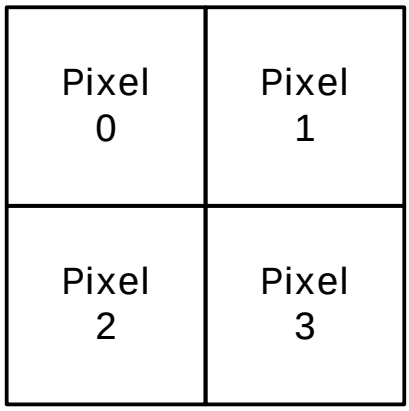
3DSTATE_SAMPLE_MASK		
	7:0	Reserved

7.3 Rasterization

The WM unit uses the setup computations performed by the SF unit to rasterize objects into the corresponding set of pixels. Most of the controls regarding the screen-space geometry of rendered objects are programmed via the SF unit.

The rasterization process generates pixels in 2x2 groups of pixels called *subspans* (see Figure 7-1) which, after being subjected to various inclusion/discard tests, are grouped and passed to spawned Pixel Shader (PS) threads for subsequent processing. Once these PS threads are spawned, the WM unit provides only bookkeeping functions on the pixels. Note that the WM unit can proceed on to rasterize subsequent objects while PS threads from previous objects are still executing.

Figure 7-1. Pixels with a SubSpan



B6850-01

7.3.1 Drawing Rectangle Clipping

The Drawing Rectangle defines the maximum extent of pixels which can be rendered. Portions of objects falling outside the Drawing Rectangle will be clipped (pixels discarded). Implementations will typically discard objects falling completely outside of the Drawing Rectangle as early in the pipeline as possible. There is no control to turn off Drawing Rectangle clipping – it is unconditional.

For the purposes of clipping, the Drawing Rectangle must itself be clipped to the destination buffer extents. (The Drawing Rectangle Origin, used to offset relative X,Y coordinates earlier in the pipeline, is permitted to lie offscreen). The **Clipped Drawing Rectangle X,Y Min,Max** state variables (programmed via 3DSTATE_DRAWING_RECTANGLE – See *SF Unit*) defines the intersection of the Drawing Rectangle and the Color Buffer. It is specified with non-negative integer pixel coordinates relative to the Destination Buffer upper-left origin.

Pixels with coordinates outside of the Drawing Rectangle cannot be rendered (i.e., the rectangle is inclusive). For example, to render to a full-screen 1280x1024 buffer, the following values would be required: Xmin=0, Ymin=0, Xmax=1279 and Ymax=1023.

For "full screen" rendering, the Drawing Rectangle coincides with the screen-sized buffer. For "front-buffer windowed" rendering it coincides with the destination "window".

7.3.2 Line Rasterization

See *SF Unit* chapter for details on the screen-space geometry of the various line types.

7.3.2.1 Coverage Values for Anti-Aliased Lines

The WM unit is provided with both the **Line Anti-Aliasing Region Width** and **Line End Cap Anti-aliasing Region Width** state variables (in WM_STATE) in order to compute the coverage values for anti-aliased lines.

7.3.2.2 3DSTATE_AA_LINE_PARAMS [DevCTG+]

3DSTATE_AA_LINE_PARAMETERS			
Project:	[DevCTG+]	**Length Bias:**	2
The 3DSTATE_AA_LINE_PARAMS command is used to specify the slope and bias terms used in the improved alpha coverage computation. Note that in these devices the coverage values passed to PS threads are full U0.8 values, versus [DevBW]/[DevCL] where U0.4 values are passed.			

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

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

The slope and bias values should be computed to closely match the reference rasterizer results. Based on empirical data, the following recommendations are offered:

The final alpha for the center of the line needs to be 148 to match the reference rasterizer. In this case, the Lo to edge 0 and edge 3 will be the same. Since the alpha for each edge is multiplied together, we get:

edge0alpha * edge1alpha = 148/255 = 0.580392157

Since edge0alpha = edge3alpha we get:

$(\text{edge0alpha})^2 = 0.580392157$

edge0alpha = sqrt(0.580392157) = 0.761834731 at the center pixel

The desired alpha for pixel 1 = 54/255 = 0.211764706

The slope is (0.761834731 – 0.211764706) = 0.550070025

Since we are using 8 bit precision, the slope becomes

AA Coverage [EndCap] Slope = 0.55078125

The alpha value for Lo = 0 (second pixel from center) determines the bias term and is equal to

(0.211764706 – 0.550070025) = -0.338305319

With 8 bits of precision the programmed bias value

AA Coverage [EndCap] Bias = 0.33984375

7.3.2.3 Line Stipple

Line stipple, controlled via the **Line Stipple Enable** state variable in WM_STATE, discards certain pixels that are produced by non-AA line rasterization.

The line stipple rule is specified via the following state variables programmed via 3DSTATE_LINE_STIPPLE: the 16-bit **Line Stipple Pattern** (p), **Line Stipple Repeat Count** l, and **Line Stipple Inverse Repeat Count.** Sofware must compute **Line Stipple Inverse Repeat Count** as 1.0f / **Line Stipple Repeat Count** and then converted from float to the required fixed point encoding (see 3STATE_LINE_STIPPLE).

The WM unit maintains an internal Line Stipple Counter state variable (s). The initial value of s is zero; s is incremented after production of each pixel of a line segment (pixels are produced in order, beginning at the starting point and working towards the ending point). s is reset to 0 whenever a new primitive is processed (unless the primitive type is LINESTRIP_CONT or LINESTRIP_CONT_BF), and before every line segment in a group of independent segments (LINELIST primitive).

During the rasterization of lines, the WM unit computes:

$$b = \lfloor s/r \rfloor \bmod 16,$$

A pixel is rendered if the bth bit of p is 1, otherwise it is discarded. The bits of p are numbered with 0 being the least significant and 15 being the most significant.

7.3.2.4 3DSTATE_LINE_STIPPLE [DevSNB]

3DSTATE_LINE_STIPPLE			
Project:	[DevSNB]	**Length Bias:**	2
The 3DSTATE_LINE_STIPPLE command is used to specify state variables used in the Line Stipple function.			
DWord	**Bit**	**Description**	
0	31:29	Command Type Default Value: 3h GFXPIPE Format: OpCode	
	28:27	Command SubType Default Value: 3h GFXPIPE_3D Format: OpCode	

3DSTATE_LINE_STIPPLE		
	26:24	3D Command Opcode Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	3D Command Sub Opcode Default Value: 08h 3DSTATE_LINE_STIPPLE Format: OpCode
	15:8	Reserved Project: All Format: MBZ
	7:0	DWord Length Default Value: 1h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31	Modify Enable (Current Repeat Counter, Current Stipple Index) Project: All Format: Enable FormatDesc Modify enable for Current Repeat Counter and Current Stipple Index fields. Programming Notes Software should never set this field to enabled. It is provided only for HW-generated commands as part of context save/restore.
	30	Reserved Project: All Format: MBZ
	29:21	Current Repeat Counter Project: All Format: U9 FormatDesc This field sets the HW-internal repeat counter state. Note: Software should never attempt to set this value – this state is only provided for HW-generated commands as part of context save/restore.
	20	Reserved Project: All Format: MBZ
	19:16	Current Stipple Index Project: All Format: U4 FormatDesc This field sets the HW-internal stipple pattern index. Note: Software should never attempt to set this value – this state is only provided for HW-generated commands as part of context save/restore.
	15:0	Line Stipple Pattern Project: All Format: 16 bit mask. Bit 15 = most significant bit, Bit 0 = least significant bit FormatDesc Specifies a pattern used to mask out bit specific pixels while rendering lines.

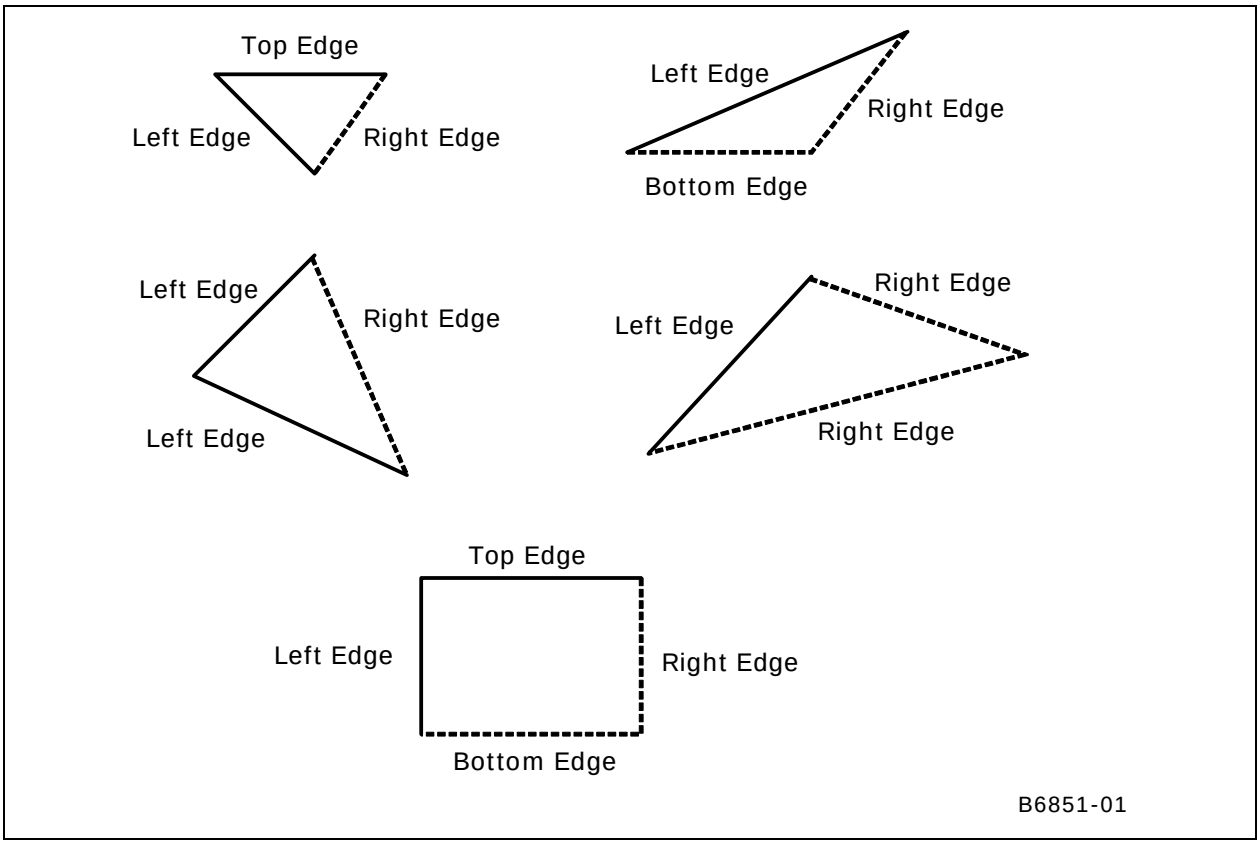
3DSTATE_LINE_STIPPLE		
2	31:16	Line Stipple Inverse Repeat Count Project: [DevSNB] Format: U1.13 FormatDesc Range [0.00390625, 1.0] Specifies the inverse (truncated) of the repeat count for the line stipple function.
	15:9	Reserved Project: All Format: MBZ
	8:0	Line Stipple Repeat Count Project: All Format: U9 FormatDesc Range [1, 256] Specifies the repeat count for the line stipple function.

7.3.3 Polygon (Triangle and Rectangle) Rasterization

The rasterization of LINE, TRIANGLE, and RECTANGLE objects into pixels requires a “pixel sampling grid” to be defined. This grid is defined as an axis-aligned array of pixel sample points spaced exactly 1 pixel unit apart. If a sample point falls within one of these objects, the pixel associated with the sample point is considered “inside” the object, and information for that pixel is generated and passed down the pipeline.

For TRIANGLE and RECTANGLE objects, if a sample point intersects an edge of the object, the associated pixel is considered “inside” the object if the intersecting edge is a “left” or “top” edge (or, more exactly, the intersected edge is not a “right” or “bottom” edge). Note that “top” and “bottom” edges are by definition exactly horizontal. The following diagram identifies the edge types for representative TRIANGLE and RECTANGLE objects (solid edges are inclusive, dashed edges are exclusive).

Figure 7-2. TRIANGLE and RECTANGLE Edge Types



7.3.3.1 Polygon Stipple

The *Polygon Stipple* function, controlled via the **Polygon Stipple Enable** state variable in WM_STATE, allows only selected pixels of a repeated 32x32 pixel pattern to be rendered. Polygon stipple is applied only to the following primitive types:

3DPRIM_POLYGON

3DPRIM_TRIFAN

3DPRIM_TRILIST

3DPRIM_TRISTRIP

3DPRIM_TRISTRIP_REVERSE

Note that the 3DPRIM_TRIFAN_NOSTIPPLE object is never subject to polygon stipple.

The stipple pattern is defined as a 32x32 bit pixel mask via the 3DSTATE_POLY_STIPPLE_PATTERN command. This is a non-pipelined command which incurs an implicit pipeline flush when executed.

The origin of the pattern is specified via **Polygon Stipple X,Y Offset** state variables programmed via the 3DSTATE_POLY_STIPPLE_OFFSET command. The offsets are pixel offsets from the Color Buffer origin to the upper left corner of the stipple pattern. This is a non-pipelined command which incurs an implicit pipeline flush when executed.

7.3.3.2 3DSTATE_POLY_STIPPLE_OFFSET

3DSTATE_POLY_STIPPLE_OFFSET			
Project:	All	**Length Bias:**	2
The 3DSTATE_POLY_STIPPLE_OFFSET command is used to specify the origin of the repeated screen-space Polygon Stipple Pattern as an X,Y offset from the Color Buffer origin.			

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

3DSTATE_POLY_STIPPLE_OFFSET		
	4:0	Polygon Stipple Y Offset Project: All Format: U5 FormatDesc Range [0,31] Specifies a 5 bit y address offset in the poly stipple pattern

7.3.3.3 3DSTATE_POLY_STIPPLE_PATTERN

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

3DSTATE_POLY_STIPPLE_PATTERN		
2..32	31:0	Polygon Stipple Pattern Rows 2-32 (bottom most) Project: All Format: 32 bit mask. Bit 31 = upper left corner, Bit 0 = upper right corner of first row. FormatDesc Specifies a pattern used by Polygon Stipple to mask out specific pixels of every 32x32 area rendered.

7.4 Multisampling [DevSNB+]

The multisampling function has two components:

- **Multisample Rasterization**: multisample rasterization occurs at a subpixel level, wherein each pixel consists of a number of "samples" at state-defined positions within the pixel footprint. Coverage of the primitive as well as color calculator operations (stencil test, depth test, color buffer blending, etc.) are done at the sample level. In addition the pixel shader itself can optionally run at the sample level depending on a separate state field.

- **Multisample Render Targets (MSRT)**: The render targets, as well as the depth and stencil buffers, now have the ability to store per-sample values. When combined with multisample rasterization, color calculator operations such as stencil test, depth test, and color buffer blending are done with the destination surface cont aining potentially different values per sample.

7.4.1 Multisample Modes/State

A number of state variables control the operation of the multisampling function. The following list indicates the state and their location. Refer to the state definition for more details.

- **Multisample Rasterization Mode** (3DSTATE_SF and 3DSTATE_WM): controls whether rasterization of non-lines is performed on a pixel or sample basis (PIXEL vs. PATTERN), and whether rasterization of lines is performed on a pixel or sample basis (OFF vs. ON). The table below details the possible values of this state:

Multisample Rasterization Mode	Description
MSRASTMODE_ OFF_PIXEL	**All object types:** Rasterization is performed on a pixel (vs. sample) basis. The number of pixel sample points is determined by **Number of Multisamples**, but the location(s) are all fixed at either the pixel center or UL corner, as defined by **Pixel Location** (3DSTATE_MULTISAMPLE). The programmed values **Sample Offset** states are ignored. **Lines:** Multisampling rasterization of lines is turned off, allowing 0-width lines, french-cut wide/stippled lines, and AA lines.
MSRASTMODE_ OFF_PATTERN	This mode is only valid when **Number of Multisamples =**

Multisample Rasterization Mode	Description
	NUMSAMPLES_4. **Non-Lines:** Rasterization is performed on a 4X sample basis. The four pixel sample points are completely defined by state variables programmed via 3DSTATE_MULTISAMPLE. **Lines:** Rasterization is performed on a pixel (vs. sample) basis. The number of pixel sample points is determined by **Number of Multisamples**, but the location(s) are all fixed at either the pixel center or UL corner, as defined by **Pixel Location** (3DSTATE_MULTISAMPLE). The programmed values **Sample Offset** states are ignored. Multisampling rasterization of lines is turned off, allowing 0-width lines, french-cut wide/stippled lines, and AA lines.
MSRASTMODE_ ON_PIXEL	**All object types:** Rasterization is performed on a pixel (vs. sample) basis. The number of pixel sample points is determined by **Number of Multisamples**, but the location(s) are all fixed at either the pixel center or UL corner, as defined by **Pixel Location** (3DSTATE_MULTISAMPLE). The programmed values **Sample Offset** states are ignored. **Lines:** Multisampling rasterization of lines is turned on, where all lines are drawn as rectangles using **Line Width**.
MSRASTMODE_ ON_PATTERN	This mode is only valid when **Number of Multisamples** = NUMSAMPLES_4. **All object types:** Rasterization is performed on a 4X sample basis. The four pixel sample points are completely defined by state variables programmed via 3DSTATE_MULTISAMPLE. **Lines:** Multisampling rasterization of lines is turned on, where lines are drawn as rectangles using **Line Width**.

- **Multisample Dispatch Mode** (3DSTATE_WM): controls whether the pixel shader is executed per pixel or per sample.
- **Number of Multisamples** (3DSTATE_MULTISAMPLE and SURFACE_STATE): indicates the number of samples per pixel contained on the surface. This field in 3DSTATE_MULTISAMPLE must match the corresponding field in SURFACE_STATE for each render target. The depth, hierarchical depth, and stencil buffers inherit this field from 3DSTATE_MULTISAMPLE.
- **Pixel Location** (3DSTATE_MULTISAMPLE): indicates the subpixel location where values specified as “pixel” are sampled. This is either the upper left corner or the center.
- **Sample Offsets** (3DSTATE_MULTISAMPLE): for each of the four samples, specifies the subpixel location of each sample.

APIs define a “Multisample” render state Boolean which controls how objects are rasterized (sample level vs. pixel level). The binding of MSRTs also affects the rasterization process. The various permutations of multisample operation are listed below, along with the HW state settings required.

API State				HW Mode
MSRT	**Sampling Pattern**	**Multisample Enable**	**PerSample PS?**	
1X	n/a	Disabled	-	Legacy Non-MSAA Mode
		Enabled	-	1X Multisampling Mode
4X	UL or Center	Disabled	No	MSRT Only, PerPixel PS
			Yes	MSRT Only, PerSample PS
		Enabled	No	Multibuffering MSAA, PerPixel PS
			Yes	Multibuffering MSAA, PerSample PS
	Pattern	Disabled	No	MSRT Only, PerPixel PS
			Yes	n/a
			No	Mixed Mode, PerPixel PS
			Yes	Mixed Mode, PerSample PS
		Enabled	No	Pattern MSAA, PerPixel PS
			Yes	Pattern MSAA, PerSample PS

HW State			HW Mode
Num Samples	**MS RAST MODE**	**MS DISP MODE**	
1X	OFF_PIXEL	PERSAMPLE	**Legacy Non-MSAA Mode** 1X rasterization, using Pixel Location Legacy lines/aa-line rasterization 1X PS, sample at Pixel Location 1X output merge, eval Depth at Pixel Location
	ON_PIXEL	PERSAMPLE	**1X Multisampling Mode** 1X rasterization, using Pixel Location MSAA lines only, using Pixel Location

HW State			HW Mode
Num Samples	**MS RAST MODE**	**MS DISP MODE**	
			1X PS, sample at Pixel Location 1X output merge, eval Depth at Pixel Location
	-	PERPIXEL	Invalid
	ON_PATTERN	-	Invalid
	OFF_PATTERN	-	Invalid
4X	OFF_PIXEL	PERPIXEL	**MSRT Only, PerPixel PS** 1X rasterization, using Pixel Location Legacy lines/aa-line rasterization 1X PS, sample at Pixel Location 4X output merge, eval Depth at Pixel Location
		PERSAMPLE	**MSRT Only, PerSample PS** 1X rasterization, using Pixel Location Legacy lines/aa-line rasterization 4X PS, all samples at Pixel Location 4X output merge, eval Depth at Pixel Location
	ON_PIXEL	PERPIXEL	**Multibuffering MSAA, PerPixel PS** 1X rasterization, using Pixel Location MSAA lines only 1X PS, sample at Pixel Location 4X output merge, eval Depth at Pixel Location
		PERSAMPLE	**Multibuffering MSAA, PerSample PS** 1X rasterization, using Pixel Location MSAA lines only 4X PS, all samples at Pixel Location 4X output merge, eval Depth at Pixel Location
	OFF_PATTERN	PERPIXEL	**Mixed Mode, PerPixel PS** Lines: Legacy lines/aa-line rasterization using Pixel Location Non-Lines: 4X rasterization, using Sample Offsets

HW State			HW Mode
Num Samples	MS RAST MODE	MS DISP MODE	
			1X PS, sample at Pixel Location 4X output merge, eval depth at Sample Offsets
		PERSAMPLE	**Mixed Mode, PerSample PS** Lines: Legacy lines/aa-line rasterization using Pixel Location Non-Lines: 4X rasterization, using Sample Offsets 4X PS, sample at Pixel Location or Sample Offsets 4X output merge, eval depth at Sample Offsets
	ON_PATTERN	PERPIXEL	**Pattern MSAA, PerPixel PS** 4X rasterization, using Sample Offsets MSAA lines only 1X PS, sample at Pixel Location 4X output merge, eval depth at Sample Offsets
		PERSAMPLE	**Pattern MSAA, PerSample PS** 4X rasterization, using Sample Offsets MSAA lines only 4X PS, sample at Pixel Location or Sample Offsets 4X output merge, eval depth at Sample Offsets

DX9 4x/8x MSAA workaround [DevSNB+]

There is hardware issue in DX9 MSAA 4x/8x mode of rendering because of which driver needs to implement a work around.

This work around requires that driver force the device to render in DX10 MSAA mode if the API mode happens to be DX9 with multi sample enabled and 4x or 8x rendertarget is bound. Workaround also requires to enable the viewport transform in SF if not enabled already and set the transform coefficients such that an offset of 0.5 gets added in horizontal and vertical directions.

Following gives the pseudocode to be implemented in DX9 mode

```
If( multisampleRenderTarget &&
   MultisampleEnable )
{
        // NDC space enabled with 3DSTATE_SF Viewport Transform Enable
```

ViewportTransformEnable = true;

// 3DSTATE_MULTISAMPLE Pixel Location to Center like DX10+

PixelLocation = 0;

// Set ViewportMatrixElements in SF_VIEWPORT

ViewportMartixElement_m00 = 1.0f;

ViewportMartixElement_m11 = 1.0f;

ViewportMartixElement_m22 = 1.0f;

ViewportMartixElement_m30 = 0.5f;

ViewportMartixElement_m31 = 0.5f;

ViewportMartixElement_m32 = 0.0f;

7.4.2 3DSTATE_MULTISAMPLE [DevSNB+]

3DSTATE_MULTISAMPLE			
Project:	[DevSNB+]	**Length Bias:**	2

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

Programming Restriction:

Driver must guarentee that all the caches in the depth pipe are flushed before this command is parsed. This requires driver to send a PIPE_CONTROL with a CS stall along with a Depth Flush prior to this command.

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

When this command is issued, the currently active depth buffer, hierarchical depth buffer, stencil buffer, and render target(s) must be cleared (meaning that every pixel must be overwritten). Alternatively, other surfaces can be activated before issuing the next 3DPRIMITIVE that were previously rendered with the same values of all state fields in this command. In other words, it is illegal to render to these surfaces with multiple different values of the state fields in this command.

DWord	Bit	Description
0	31:29	Command Type Default Value: 3h GFXPIPE Format: OpCode

3DSTATE_MULTISAMPLE

	28:27	Command SubType Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	3D Command Opcode Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	3D Command Sub Opcode Default Value: 0Dh 3DSTATE_MULTISAMPLE Format: OpCode
	15:8	Reserved Project: All Format: MBZ
	7:0	DWord Length Default Value: 1h [DevSNB] Excludes DWord (0,1) 2h [Reserved] Format: =n Total Length - 2 Project: All
1	31:6	Reserved Project: All Format: MBZ
	5	**Reserved**
	4	**Pixel Location** Project: All Format: U1 Enumerated Type FormatDesc This field specifies where the device evaluates “pixel” (vs. centroid or sample) values/attributes.

Value	Name	Description	Project
0h	PIXLOC_CENTER	Use the pixel center (0.5, 0.5 offset)	All
1h	PIXLOC_UL_CORNER	Use the pixel upper-left corner	All

Programming Notes
The programming of this field is assumed to be a function of the API being supported. Specifically, it is expected that OpenGL API requires CENTER selection,.

3DSTATE_MULTISAMPLE

	3:1	Number of Multisamples Project: All Format: U3 Enumerated Type FormatDesc This field specifies how many samples/pixel exist in all RTs and the Depth Buffer, as log2(#samples). This field is valid regardless of the setting of **Multisample Rasterization Mode**.
	0	Reserved Project: All Format: MBZ

Value	Name	Description	Project
0h	NUMSAMPLES_1	1 sample/pixel	All
1h	Reserved		All
2h	NUMSAMPLES_4	4 samples/pixel	All
3h	Reserved		All
4h-7h	Reserved		All

Programming Notes
Setting **Multisample Rasterization Mode** to MSRASTMODE_xxx_PATTERN when **Number of Multisamples** == NUMSAMPLES_1 is UNDEFINED.
The setting of this field must match the **Number of Multisamples** field in SURFACE_STATE of all bound render targets.

3DSTATE_MULTISAMPLE		
2	31:28	Sample3 X Offset Project: All Format: U0.4 FormatDesc Range [0,0.9375] Subpixel X offset of Sample 3 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.
	27:24	Sample3 Y Offset Project: All Format: U0.4 FormatDesc Range [0,0.9375] Subpixel Y offset of Sample 3 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.
	23:20	Sample2 X Offset Project: All Format: U0.4 FormatDesc Range [0,0.9375] Subpixel X offset of Sample 2 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.
	19:16	Sample2 Y Offset Project: All Format: U0.4 FormatDesc Range [0,0.9375] Subpixel Y offset of Sample 2 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.
	15:12	Sample1 X Offset Project: All Format: U0.4 FormatDesc Range [0,0.9375] Subpixel X offset of Sample 1 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.

3DSTATE_MULTISAMPLE		
	11:8	Sample1 Y Offset Project: All Format: U0.4 FormatDesc Range [0,0.9375] Subpixel Y offset of Sample 1 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.
	7:4	Sample0 X Offset Project: All Format: U0.4 FormatDesc Range [0,0.9375] Subpixel X offset of Sample 0 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.
	3:0	Sample0 Y Offset Project: All Format: U0.4 FormatDesc Range [0,0.9375] Subpixel Y offset of Sample 0 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.
3	31:28	Reserved
	27:24	Reserved
	23:20	Reserved
	19:1	Reserved
	15:12	Reserved
	11:8	Reserved
	7:4	Reserved
	3:0	Reserved

7.5 Early Depth/Stencil Processing

The Windower/IZ unit provides the Early Depth Test function, a major performance-optimization feature where an attempt is made to remove pixels that fail the Depth and Stencil Tests prior to pixel shading. This requires the WM unit to perform the interpolation of pixel ("source") depth values, read the current ("destination") depth values from the cached depth buffer, and perform the Depth and Stencil Tests. As the WM unit has per-pixel source and destination Z values, these values are passed in the PS thread payload, if required.

7.5.1 Depth Offset

Note for [DevSNB+]: the depth offset function is contained in SF unit, thus the state to control it is also contained in SF unit.

There are occasions where the Z position of some objects need to be slightly offset in order to reduce artifacts due to coplanar or near-coplanar primitives. A typical example is drawing the edges of triangles as wireframes – the lines need to be drawn slightly closer to the viewer to ensure they will not be occluded by the underlying polygon. Another example is drawing objects on a wall – without a bias on the z positions, they might be fully or partially occluded by the wall.

The device supports *global* depth offset, applied only to triangles, that bases the offset on the object's z slope. Note that there is no clamping applied at this stage after the Z position is offset – clamping to [0,1] can be performed later after the Z position is interpolated to the pixel. This is preferable to clamping prior to interpolation, as the clamping would change the Z slope of the entire object.

The Global Depth Offset function is controlled by the **Global Depth Offset Enable** state variable in WM_STATE. Global Depth Offset is only applied to 3DOBJ_TRIANGLE objects.

When Global Depth Offset Enable is ENABLED, the pipeline will compute:

MaxDepthSlope = max(abs(dZ/dX),abs(dz/dy)) // approximation of max depth slope for polygon

When UNORM Depth Buffer is at Output Merger (or no Depth Buffer):

```
Bias = GlobalDepthOffsetConstant * r + GlobalDepthOffsetScale * MaxDepthSlope
```

Where r is the minimum representable value > 0 in the depth buffer format, converted to float32. (note: If state bit **Legacy Global Depth Bias Enable** is set, the r term will be forced to 1.0)

When Floating Point Depth Buffer at Output Merger:

```
Bias = GlobalDepthOffsetConstant * 2^(exponent(max z in primitive) - r) +
        GlobalDepthOffsetScale * MaxDepthSlope
```

Where r is the # of mantissa bits in the floating point representation (excluding the hidden bit), e.g. 23 for float32. (note: If state bit Legacy Global Depth Bias Enable is set, no scaling is applied to the GobalDepthOffsetConstant).

Adding Bias to z:

```
 if (GlobalDepthOffsetClamp > 0)

        Bias = min(DepthBiasClamp, Bias)

else if(GlobalDepthOffsetClamp < 0)

        Bias = max(DepthBiasClamp, Bias)
```

```
// else if GlobalDepthOffsetClamp == 0, no clamping occurs

z = z + Bias
```

Biasing is constant for a given primitive. The biasing formulas are performed with float32 arithmetic. Global Depth Bias is not applied to any point or line primitives

7.5.2 Early Depth Test/Stencil Test/Write

When **Early Depth Test Enable** is ENABLED, the WM unit will attempt to discard depth-occluded pixels during scan conversion (before processing them in the Pixel Shader). Pixels are only discarded when the WM unit can ensure that they would have no impact to the ColorBuffer or DepthBuffer. This function is therefore only a performance feature. **Note:** for **[DevSNB+]**, the **Early Depth Test Enable** bit is no longer present. This function is always enabled.

If some pixels within a subspan are discarded, only the pixel mask is affected indicating that the discarded pixels are not active. If all pixels within a subspan are discarded, that subspan will not even be dispatched.

7.5.2.1 Software-Provided PS Kernel Info

In order for the WM unit to properly perform Early Depth Test and supply the proper information in the PS thread payload (and even determine if a PS thread needs to be dispatched), it requires information regarding the PS kernel operation. This information is provided by a number of state bits in WM_STATE, as summarized in the following table.

State Bit	Description
Pixel Shader Kill Pixel	This must be set when there is a chance that valid pixels passed to a PS thread may be discarded. This includes the discard of pixels by the PS thread resulting from a “killpixel” or “alphatest” function or as dictated by the results of the sampling of a “chroma-keyed” texture. The WM unit needs this information to prevent early depth/stencil writes for pixels which might be killed by the PS thread, etc. See WM_STATE/3DSTATE_WM for more information.
Pixel Shader Computed Depth	This must be set when the PS thread computes the “source” depth value (i.e., from the API POV, writes to the “oDepth” output). In this case the WM unit can’t make any decisions based on the WM-interpolated depth value. See WM_STATE/3DSTATE_WM for more information.

State Bit	Description
Pixel Shader Uses Source Depth	Must be set if the PS thread requires the WM-interpolated source depth value. This will force the source depth to be passed in the thread payload where otherwise the WM unit would not have seen it as required. See WM_STATE/3DSTATE_WM for more information.

7.5.2.2 Early Depth Test Cases

Note: for **[DevSNB+]**, the functional details of the early depth test feature are not visible to software.

7.5.3 Hierarchical Depth Buffer

A hierarchical depth buffer is supported to reduce memory traffic due to depth buffer accesses. This buffer is supported only in Tile Y memory.

The **Surface Type**, **Height**, **Width**, **Depth**, **Minimum Array Element**, **Render Target View Extent**, and **Depth Coordinate Offset X/Y** of the hierarchical depth buffer are inherited from the depth buffer. The height and width of the hierarchical depth buffer that must be allocated are computed by the following formulas, where HZ is the hierarchical depth buffer and Z is the depth buffer. The Z_Height, Z_Width, and Z_Depth values given in these formulas are those present in 3DSTATE_DEPTH_BUFFER incremented by one. **[DevSNB+]**: The value of Z_Height and Z_Width must each be multiplied by 2 before being applied to the table below if **Number of Multisamples** is set to NUMSAMPLES_4. The value of Z_Height must be multiplied by 2 and Z_Width must be multiplied by 4 before being applied to the table below if **Number of Multisamples** is set to NUMSAMPLES_8.

[DevSNB]

Surface Type	HZ_Width (bytes)	HZ_Height (rows)
SURFTYPE_1D	ceiling(Z_Width / 16) * 16	4 * Z_Depth
SURFTYPE_2D	ceiling(Z_Width / 16) * 16	ceiling(Z_Height / 8) * 4 * Z_Depth
SURFTYPE_3D	ceiling(Z_Width / 16) * 16	ceiling(Z_Height / 8) * 4 * Z_Depth
SURFTYPE_CUBE	ceiling(Z_Width / 16) * 16	ceiling(Z_Height / 8) * 24 * Z_Depth

[DevSNB]: The hierarchical depth buffer does not support the **LOD** field, it is assumed by hardware to be zero. A separate hierarachical depth buffer is required for each LOD used, and the corresponding buffer's state delivered to hardware each time a new depth buffer state with modified LOD is delivered.

If HiZ is enabled, you must initialize the clear value by either

a. Perform a depth clear pass to initialize the clear value.

b. Send a 3dstate_clear_params packet with valid = 1

Without one of these events, context switching will fail, as it will try to save off a clear value even though no valid clear value has been set. When context restore happens, HW will restore an uninitialized clear value.

Erratum: [DevSNB:GT1:P0]:

1. Hierarchical Depth Buffer must not be enabled while rendering with 3DSTATE_WM.RAST_MODE = *_*_PATTERN and Barycentric Interpolation Mode is set to Perspective Centroid or Non-perspective Centroid when NUM_MULTISAMPLES = 4.

2. Hierarchical Depth Buffer must not be enabled while rendering with 3DSTATE_WM.RAST_MODE = *_*_PATTERN and Position ZW Interpolation Mode is set to centroid when NUM_MULTISAMPLES = 4 and PS_USE_SOURCE_Z/W.

3. Hierarchical Depth Buffer must not be enabled while rendering with 3DSTATE_WM.RAST_MODE = *_*_PATTERN and Position XY Offset Select set to Centroid when NUM_MULTISAMPLES = 4.

7.5.3.1 Depth Buffer Clear

With the hierarchical depth buffer enabled, performance is generally improved by using the special clear mechanism described here to clear the hierarchical depth buffer and the depth buffer. This is enabled though the **Depth Buffer Clear** field in WM_STATE or 3DSTATE_WM. This bit can be used to clear the depth buffer in the following situations:

- Complete depth buffer clear
- Partial depth buffer clear with the clear value the same as the one used on the previous clear
- Partial depth buffer clear with the clear value different than the one used on the previous clear can use this mechanism if a depth buffer resolve is performed first.

The following is required when performing a depth buffer clear with this field:

- If other rendering operations have preceded this clear, a PIPE_CONTROL with write cache flush enabled and Z-inhibit disabled must be issued before the rectangle primitive used for the depth buffer clear operation.
- The fields in 3DSTATE_CLEAR_PARAMS are set to indicate the source of the clear value and (if source is in this command) the clear value itself.
- A rectangle primitive representing the clear area is delivered. The primitive must adhere to the following restrictions on size:
    - If **Number of Multisamples** is NUMSAMPLES_1, the rectangle must be aligned to an 8x4 pixel block relative to the upper left corner of the depth buffer, and contain an integer number of these pixel blocks, and all 8x4 pixels must be lit.
    - If **Number of Multisamples** is NUMSAMPLES_4, the rectangle must be aligned to a 4x2 pixel block (8x4 sample block) relative to the upper left corner of the depth buffer, and contain an integer number of these pixel blocks, and all samples of the 4x2 pixels must be lit
    - If **Number of Multisamples** is NUMSAMPLES_8, the rectangle must be aligned to a 2x2 pixel block (8x4 sample block) relative to the upper left corner of the depth buffer, and contain an integer number of these pixel blocks, and all samples of the 2x2 pixels must be list.
- **Depth Test Enable** must be disabled and **Depth Buffer Write Enable** must be enabled (if depth is being cleared).
- Stencil buffer clear can be performed at the same time by enabling Stencil Buffer Write Enable. Stencil Test Enable must be enabled and Stencil Pass Depth Pass Op set to REPLACE, and the clear value that is placed in the stencil buffer is the **Stencil Reference Value** from COLOR_CALC_STATE.

- Note also that stencil buffer clear can be performed without depth buffer clear. For stencil only clear, **Depth Test Enable** and **Depth Buffer Write Enable** must be disabled.
- **[DevSNB]** errata: For stencil buffer only clear, the previous depth clear value must be delivered during the clear.
- **Pixel Shader Dispatch, Alpha Test, Pixel Shader Kill Pixel** and **Pixel Shader Computed Depth** must all be disabled.

ILK: Several cases exist where **Depth Buffer Clear** with Fast Clear Optimization enabled (Cache Mode Register offset 0x**2120, bit 2**) cannot be enabled:

- If the depth buffer format is D32_FLOAT_S8X24_UINT or D24_UNORM_S8_UINT.
- If the separate stencil buffer is disabled.

[DevSNB+]: Several cases exist where **Depth Buffer Clear** cannot be enabled (the legacy method of clearing must be performed):

- If the depth buffer format is D32_FLOAT_S8X24_UINT or D24_UNORM_S8_UINT.
- If stencil test is enabled but the separate stencil buffer is disabled.
- [DevSNB-A{W/A}]: When fast clear optimization is enabled, depth buffer clear pass must have a rectangle aligned to 8X4 pixel block. Further the Height and Width of the clear rectangle must be a multiple of 8 pixels and 4 lines, respectively.
- [DevSNB{W/A}]: When depth buffer format is D16_UNORM and the width of the map (LOD0) is not multiple of 16, fast clear optimization must be disabled.
- [DevSNB, DevSNB-B{W/A}]: Depth buffer clear pass must be followed by a PIPE_CONTROL command with DEPTH_STALL bit set and Then followed by Depth FLUSH

7.5.3.2 Depth Buffer Resolve

If the hierarchical depth buffer is enabled, the depth buffer may contain incorrect results after rendering is complete. If the depth buffer is retained and used for another purpose (i.e as input to the sampling engine as a shadow map), it must first be “resolved”. This is done by setting the **Depth Buffer Resolve Enable** field in WM_STATE or 3DSTATE_WM and rendering a full render target sized rectangle. Once this is complete, the depth buffer will contain the same contents as it would have had the rendering been performed with the hierarchical depth buffer disabled. In a typical usage model, depth buffer needs to be resolved after rendering on it and before using a depth buffer as a source for any consecutive operation. Depth buffer can be used as a source in three different cases: using it as a texture for the nest rendering sequence, honoring a lock on the depth buffer to the host OR using the depth buffer as a blit source.

The following is required when performing a depth buffer resolve:

- A rectangle primitive of the same size as the previous depth buffer clear operation must be delivered, and depth buffer state cannot have changed since the previous depth buffer clear operation.
- **Depth Test Enable** must be enabled with the **Depth Test Function** set to NEVER. **Depth Buffer Write Enable** must be enabled. **Stencil Test Enable** and **Stencil Buffer Write Enable** must be disabled.
- **Pixel Shader Dispatch, Alpha Test, Pixel Shader Kill Pixel** and **Pixel Shader Computed Depth** must all be disabled.

- [DevSNB-A]: When fast clear optimization is enabled, depth buffer resolve pass must have a rectangle aligned to 8X4 pixel block. Further, the Height and Width of the clear rectangle must be a multiple of 8 pixels and 4 lines, respectively.

7.5.3.3 Hierarchical Depth Buffer Resolve

If the hierarchical depth buffer is enabled, the hierarchical depth buffer may contain incorrect results if the depth buffer is written to outside of the 3D rendering operation. If this occurs, the hierarchical depth buffer must be "resolved"to avoid incorrect device behavior. This is done by setting the Hierarchical Depth Buffer Resolve Enable field in WM_STATE or 3DSTATE_WM and rendering a full render target sized rectangle. Once this is complete, the hierarchical depth buffer will contain contents such that rendering will give the same results as it would have had the rendering been performed with the hierarchical depth buffer disabled.

The following is required when performing a hierarchical depth buffer resolve:

- A rectangle primitive covering the full render target must be delivered.
- **Depth Test Enable** must be disabled. **Depth Buffer Write Enable** must be enabled. **Stencil Test Enable** and **Stencil Buffer Write Enable** must be disabled.
- **Pixel Shader Dispatch, Alpha Test, Pixel Shader Kill Pixel** and **Pixel Shader Computed Depth** must all be disabled.

Errata: [DevSNB:A0]:

- If alpha-test or kill-pix or computed depth is enabled and stencil test or write is enabled, Hierarchical Depth Buffer must be disabled.

Errata: [DevSNB-A,B,C]:

- Hierarchical Depth Buffer must be disabled in the follwing condition:
  Anti Alias Lines are enabled.

7.5.3.4 Optimized Hierarchical Depth Buffer Resolve

If the hierarchical depth buffer is enabled, the hierarchical depth buffer may contain incorrect results if the depth buffer is written to outside of the 3D rendering operation. If this occurs, the hierarchical depth buffer must be "resolved" to avoid incorrect device behavior. This is done by setting the **Hierarchical Depth Buffer Resolve Enable** field in 3DSTATE_WM_HZ_OP and specifying a full render target sized rectangle. The depth buffer resolve uses the same sequence as the optimized Depth buffer clear (see above) except the **Hierarchical Depth Buffer Resolve Enable** bit is set. Once this is complete, the hierarchical depth buffer will contain contents such that rendering will give the same results as it would have had the rendering been performed with the hierarchical depth buffer disabled.

The following is required when performing a hierarchical depth buffer resolve:

- A rectangle primitive covering the full render target must be delivered.

7.5.4 Separate Stencil Buffer

7.5.4.1 Separate Stencil Buffer [DevSNB]

A separate stencil buffer is supported, which improves performance when using the hierarchical depth buffer with stencil test enabled. This buffer is supported only in Tile Y memory. If the separate stencil buffer is enabled, it always has the format S8_UINT. The **Surface Type**, **Height**, **Width**, and **Depth**, **Minimum Array Element**, **Render Target View Extent**, and **Depth Coordinate Offset X/Y** of the stencil buffer are inherited from the depth buffer.

The stencil depth buffer does not support the **LOD** field, it is assumed by hardware to be zero. A separate stencil depth buffer is required for each LOD used, and the corresponding buffer's state delivered to hardware each time a new depth buffer state with modified LOD is delivered.

The stencil channel in the depth buffer is still supported, however if this is used with the hierarchical depth buffer, performance will generally be lower than using the separate stencil buffer.

7.5.5 Depth/Stencil Buffer State

7.5.5.1 3DSTATE_DEPTH_BUFFER

7.5.5.1.1 3DSTATE_DEPTH_BUFFER [DevSNB]

3DSTATE_DEPTH_BUFFER					
Project:	All			**Length Bias:**	2
The depth buffer surface state is delivered as a non-pipelined state packet.					
DWord	**Bit**	**Description**			
0	31:29	Command Type Default Value: 3h GFXPIPE Format: OpCode			
	28:27	Command SubType Default Value: 3h GFXPIPE_3D Format: OpCode			
	26:24	3D Command Opcode Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode			
	23:16	3D Command Sub Opcode Default Value: 05h 3DSTATE_DEPTH_BUFFER Format: OpCode			

3DSTATE_DEPTH_BUFFER

	7:0	DWord Length Default Value: 5h Excludes DWord (0,1) Format: =n Total Length - 2 Project: [DevSNB+]
1	31:29	Surface Type Project: All Format: U3 Enumerated Type This field defines the type of the surface.
	28	Reserved Project: All Format: MBZ

Value	Name	Description	Project
0h	SURFTYPE_1D	Defines a 1-dimensional map or array of maps	All
1h	SURFTYPE_2D	Defines a 2-dimensional map or array of maps	All
2h	SURFTYPE_3D	Defines a 3-dimensional (volumetric) map	All
3h	SURFTYPE_CUBE	Defines a cube map	All
4h-6h	Reserved		All
7h	SURFTYPE_NULL	Defines a null surface	All

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

3DSTATE_DEPTH_BUFFER

27

Tiled Surface

Project: All

Format: U1 enumerated type FormatDesc

Specifies if the surface is tiled.

Value	Name	Description	Project
0h	FALSE	Linear surface	All
1h	TRUE	Tiled surface	All

Programming Notes	Project
Linear surfaces can be mapped to Main Memory (uncached) or System Memory (cacheable, snooped). Tiled surfaces can only be mapped to Main Memory. [DevILK+] : When Hierarchical Depth Buffer is enabled, this bit must be set. **[DevSNB+]:** This field must be set to TRUE. Linear Depth Buffer is not supported.	All
The corresponding cache(s) must be invalidated before a previously accessed surface is accessed again with an altered state of this bit.	All

26

Tile Walk

Project: All

Format: U1 enumerated type FormatDesc

This field specifies the type of memory tiling (XMajor or YMajor) employed to tile this surface. The Depth Buffer, if tiled, must use Y-Major tiling. See *Memory Interface Functions* for details on memory tiling and restrictions.

This field is ignored when the surface is linear.

Value	Name	Description	Project
0h	Reserved		All
1h	TILEWALK_YMAJOR	Y major tiled	All

	25	Depth Buffer Coordinate Offset Disable Project: [Pre-DevCTG] Format: Disable FormatDesc Disables the application (addition) of the “upper bits” of the Drawing Rectangle Origin to Depth Buffer coordinates. (This does not affect the application of the Drawing Rectangle Origin to the Color Buffer coordinates). This control is provided to better support “Front Buffer Rendering”. By disabling the Draw Rectangle adjustment of Depth Buffer coordinates, software can utilize a “window-sized” Depth Buffer while rendering to a window within the Color Buffer. Without this control, use of the Draw Rectangle adjustment would require the Depth Buffer to be dimensioned to match the Color Buffer (screen) vs. the target window.

Programming Notes	Project
The device still applies some small coordinate offset in order to provide the required alignment of color and depth memory/cache accesses. Software needs to consider this alignment when allocating depth buffers.	All
This bit must not be set when rendering to field-mode (interlaced) Color Buffers (i.e., when Surface State’s VerticalLineStride==1).	All
This bit can only be set when rendering to surfaces of type SURFTYPE_1D and SURFTYPE_2D with **Depth** = 0 (non-array) and **LOD** = 0 (non-mip mapped)	All

	25	Reserved Project: [DevCTG+] Format: MBZ

	24:23	Software Tiled Rendering Mode Project: All Format: U2 enumerated type FormatDesc This field is intended to enable *software tiled rendering (STR)*. If certain restrictions are met, performance can be improved by reducing memory bandwidth to the render target and depth buffer. **Normal mode:** Rendering behaves normally. **STR1 mode:** Only pixels within a particular 64x32 block (aligned relative to the upper left corner of the render target) are rendered between pixel shader serializations. Generally the alignment is guaranteed via a scissor rectangle. A write to a given pixel in the render target must occur before a read from the same pixel. **STR2 mode:** The restrictions of STR1 mode applies, and in addition each pixel must be rendered with depth write enabled and depth test disabled before it can be rendered with depth test enabled. The depth buffer in memory is not updated, even on a render cache flush. Depth buffer data is contained only within the render cache during rendering.

Value	Name	Description	Project
0h	NORMAL	Normal mode	All
1h	STR1	STR1 mode	[DevCTG+]
2h	Reserved		All
3h	STR2	STR2 mode	[DevCTG+]

Programming Notes	**Project**
[DevSNB]: Only mode is supported	
The render cache must be flushed when this field is modified from its previous state	All
For both STR modes, the depth buffer (if used) must be tiled Y with D16_UNORM format, and the render target surface must be tiled X or Y	All
For both STR modes, the only data port messages allowed that use the render cache are the Render Target UNORM Read and Write messages.	All
Performance considerations: Both STR modes eliminate all memory read traffic from the render target. The STR2 mode additionally eliminates all memory traffic to the depth buffer.	All
When STR2 mode is used in conjunction with the multi-context scheduler, context switches can only occur on the boundaries between the 64x32 blocks, as the depth buffer contents are not saved for restore when the context is restarted.	All
This field must be set to NORMAL if the **Render Cache Operational Flush Enable** bit is enabled in the Cache_Mode_0 register.	All

3DSTATE_DEPTH_BUFFER

22 — Hierarchical Depth Buffer Enable

Project: [DevILK+]

Format: Enable FormatDesc

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

Programming Notes
If this field is enabled, the **Software Tiled Rendering Mode** must be NORMAL.
This field must be disabled if **Early Depth Test Enable** is disabled.
[DevSNB+]: This field must be disabled if **Anti-aliasing Enable** in 3DSTATE_SF is enabled.
[DevSNB+]: If this field is enabled, the **Surface Format** of the depth buffer cannot be D32_FLOAT_S8X24_UINT or D24_UNORM_S8_UINT. Use of stencil requires the separate stencil buffer.

21 — Separate Stencil Buffer Enable

Project: [DevILK+]

Format: Enable FormatDesc

If enabled, indicates that a separate stencil buffer is defined.

Programming Notes
[DevSNB]: This field must be set to the same value (enabled or disabled) as Hierarchical Depth Buffer Enable.

20:18 — Surface Format

Project: All

Format: U3 enumerated type FormatDesc

Specifies the format of the depth buffer. See the **Separate Stencil Buffer Enable** and **Hierarchical Depth Buffer Enable** fields for restrictions on the use of some of these formats.

Erratum: [DevSNB-A]: Depth format:D32_FLOAT_S8X24_UINT is not supported when NUM_SAMPLES = 4 and PS_KILL_PIX bit or PS Computed Depth bit is set in the WM state.

Value	Name	Description	Project
0h	D32_FLOAT_S8X24_UINT	D32_FLOAT_S8X24_UINT	All
1h	D32_FLOAT	D32_FLOAT	All
2h	D24_UNORM_S8_UINT	D24_UNORM_S8_UINT	All
3h	D24_UNORM_X8_UINT	D24_UNORM_X8_UINT	[DevILK+]
4h	Reserved	Reserved	All
5h	D16_UNORM	D16_UNORM	All
6h-7h	Reserved	Reserved	All

3DSTATE_DEPTH_BUFFER

	17	Reserved Project: All Format: MBZ
	16:0	Surface Pitch Project: All Format: U17 pitch in (Bytes – 1) FormatDesc Range if linear: [63, 128K-1] corresponding to [64B, 128KB] also restricted to a multiple of 64B if tiled: [127, 128K-1] corresponding to [128B, 128KB] also restricted to a multiple of 128B This field specifies the pitch of the depth buffer in (#Bytes – 1).

Programming Notes	Project
If this surface is tiled, the pitch specified must be a multiple of the tile pitch, in the range [128B, 128KB].	All
If the surface is linear, the pitch can be any multiple of 64 bytes up to 128KB.	All

2	31:0	Surface Base Address Project: All Address: GraphicsAddress[31:0] Surface Type: Depth Buffer This field specifies the starting DWord address of the buffer in mapped Graphics Memory.

Programming Notes
The Depth Buffer can only be mapped to Main Memory (uncached).
If the surface is tiled, the base address must conform to the Per-Surface Tiling Alignment Rules.
If the buffer is linear, the surface must be 64-byte aligned.

3DSTATE_DEPTH_BUFFER

3	31:19	Height Project: All Format: U13 FormatDesc Range SURFTYPE_1D: must be zero SURFTYPE_2D: height of surface – 1 (y/v dimension) [0,8191] SURFTYPE_3D: height of surface – 1 (y/v dimension) [0,2047] SURFTYPE_CUBE: height of surface – 1 (y/v dimension) [0,8191] This field specifies the height of the surface. If the surface is MIP-mapped, this field contains the height of the base MIP level. **Programming Notes** The **Height** of the depth buffer must be the same as the **Height** of the render target(s) (defined in SURFACE_STATE), unless **Surface Type** is SURFTYPE_1D or SURFTYPE_2D with **Depth** = 0 (non-array) and **LOD** = 0 (non-mip mapped).
	18:6	Width Project: All Format: U13 FormatDesc Range SURFTYPE_1D: width of surface – 1 (x/u dimension) [0,8191] SURFTYPE_2D: width of surface – 1 (x/u dimension) [0,8191] SURFTYPE_3D: width of surface – 1 (x/u dimension) [0,2047] SURFTYPE_CUBE: width of surface – 1 (x/u dimension) [0,8191] This field specifies the width of the surface. If the surface is MIP-mapped, this field specifies the width of the base MIP level. The width is specified in units of pixels.

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

3DSTATE_DEPTH_BUFFER

5:2

LOD

Project: All

Format: U4 in LOD units FormatDesc

Range [0, 13]

This field defines the MIP level that is currently being rendered into.

Programming Notes	Project
The **LOD** of the depth buffer must be the same as the **LOD** of the render target(s) (defined in SURFACE_STATE).	All

1

MIP Map Layout Mode

Project: All

Format: U1 enumerated type FormatDesc

For 1D and 2D Surfaces:

This field specifies which MIP map layout mode is used, whether the map for LOD 1 is stored to the right of the LOD 0 map, or stored below it. See Memory Data Formats for details on the specifics of each layout mode.

For Other Surfaces:
This field is reserved : MBZ

Value	Name	Description	Project
0h	MIPLAYOUT_BELOW	MIPLAYOUT_BELOW	All
1h	MIPLAYOUT_RIGHT	MIPLAYOUT_RIGHT	All

Programming Notes	**Project**
MIPLAYOUT_RIGHT is legal only for 2D non-array surfaces	All

0

Reserved Project: All Format: MBZ

3DSTATE_DEPTH_BUFFER		
4	31:21	Depth Project: All Format: U11 FormatDesc Range SURFTYPE_1D: number of array elements – 1 [0,511] SURFTYPE_2D: number of array elements – 1 [0,511] SURFTYPE_3D: depth of surface – 1 (r/z dimension) [0,2047] SURFTYPE_CUBE: must be zero This field specifies the total number of levels for a volume texture or the number of array elements allowed to be accessed starting at the **Minimum Array Element** for arrayed surfaces. If the volume texture is MIP-mapped, this field specifies the depth of the base MIP level. Programming Notes The **Depth** of the depth buffer must be the same as the **Depth** of the render target(s) (defined in SURFACE_STATE).
	20:10	Minimum Array Element Project: All Format: U11 FormatDesc Range SURFTYPE_1D/2D: [0,511] SURFTYPE_3D: [0,2047] For 1D and 2D Surfaces: This field indicates the minimum array element that can be accessed as part of this surface. The delivered array index is added to this field before being used to address the surface. For 3D Surfaces: This field indicates the minimum 'R' coordinate on the LOD currently being rendered to. This field is added to the delivered array index before it is used to address the surface. For Other Surfaces: This field is ignored.

	9:1	Render Target View Extent Project: All Format: U9 FormatDesc Range SURFTYPE_1D/2D: same value as Depth field SURFTYPE_3D: [0,511] to indicate extent of [1,512] For 3D Surfaces: This field indicates the extent of the accessible 'R' coordinates minus 1 on the LOD currently being rendered to. For 1D and 2D Surfaces: This field must be set to the same value as the **Depth** field. For Other Surfaces: This field is ignored.
	0	Reserved Project: All Format: MBZ
5	31:16	Depth Coordinate Offset Y Project: [DevCTG+] Format: S15 in Screen Space (pixels) FormatDesc (3 LSBs MBZ) Range [-8192,8191] (Bits 31:30 should be a sign extension) Specifies a signed pixel offset to be added to the RenderTarget Y coordinate in order to generate a DepthBuffer Y coordinate. (See Depth Coordinate in Windower).

Programming Notes
The 3 LSBs of both offsets must be zero to ensure correct alignment
Software must ensure that the resulting (sum) coordinate value is non-negative.
This field must be zero when rendering to field-mode (interlaced) Color Buffers (i.e., when Surface State's VerticalLineStride==1).
This field can only be nonzero when rendering to surfaces of type SURFTYPE_1D and SURFTYPE_2D with Depth = 0 (non-array) and LOD = 0 (non-mip mapped)
[DevSNB-A]: This field must be zero when separate stencil buffer is enabled.

3DSTATE_DEPTH_BUFFER

	15:0	Depth Coordinate Offset X Project: [DevCTG+] Format: S15 in Screen Space (pixels) FormatDesc (3 LSBs MBZ) Range [-8192,8191] (Bits 15:14 should be a sign extension) Specifies a signed pixel offset to be added to the RenderTarget X coordinate in order to generate a DepthBuffer X coordinate. (See Depth Coordinate in Windower). **Programming Notes** The 3 LSBs of both offsets must be zero to ensure correct alignment Software must ensure that the resulting (sum) coordinate value is non-negative. This field must be zero when rendering to field-mode (interlaced) Color Buffers (i.e., when Surface State's VerticalLineStride==1). This field can only be nonzero when rendering to surfaces of type SURFTYPE_1D and SURFTYPE_2D with Depth = 0 (non-array) and LOD = 0 (non-mip mapped) [DevSNB-A]: This field must be zero when separate stencil buffer is enabled.
6	31:27	Depth Buffer Object Control State Project: All Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for the depth buffer.
	26:0	Reserved Project: All Format: MBZ

7.5.5.2 3DSTATE_STENCIL_BUFFER

7.5.5.2.1 3DSTATE_STENCIL_BUFFER [DevSNB]

3DSTATE_STENCIL_BUFFER			
Project:	[DevSNB]	**Length Bias:**	2
This command sets the surface state of the separate stencil buffer, delivered as a non-pipelined state command..			

DWord	Bit	Description
0	31:29	Command Type Default Value: 3h GFXPIPE Format: OpCode
	28:27	Command SubType Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	3D Command Opcode Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	3D Command Sub Opcode Default Value: 0Eh 3DSTATE_STENCIL_BUFFER Format: OpCode
	15:8	Reserved Project: All Format: MBZ
	7:0	DWord Length Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:29	Reserved Project: All Format: MBZ
	28:25	Stencil Buffer Object Control State Project: [DevSNB] Format: MEMORY_OBJECT_CONTROL_STATE FormatDesc Specifies the memory object control state for the stencil buffer.
	24:17	Reserved Project: All Format: MBZ

3DSTATE_STENCIL_BUFFER

	16:0	Surface Pitch Project: All Format: U17 pitch in (Bytes – 1) FormatDesc Range [127, 128K-1] corresponding to [128B, 128KB] also restricted to a multiple of 128B This field specifies the pitch of the stencil buffer in (#Bytes – 1). Programming Notes — Project Since this surface is tiled, the pitch specified must be a multiple of the tile pitch, in the range [128B, 128KB]. — All The pitch must be set to 2x the value computed based on width, as the stencil buffer is stored with two rows interleaved. Refer to "Memory Data Formats" chapter for details on the separate stencil buffer storage format in memory. — [DevSNB]
2	31:0	Surface Base Address Project: All Address: GraphicsAddress[31:0] Surface Type: Stencil Buffer This field specifies the starting DWord address of the buffer in mapped Graphics Memory. Programming Notes The Stencil Buffer can only be mapped to Main Memory (uncached). Since this surface is tiled, the base address must conform to the Per-Surface Tiling Alignment Rules.

7.5.5.3 3DSTATE_HIER_DEPTH_BUFFER

7.5.5.3.1 3DSTATE_HIER_DEPTH_BUFFER [DevSNB]

3DSTATE_HIER_DEPTH_BUFFER			
Project:	[DevSNB]	**Length Bias:**	2
This command sets the surface state of the hierarchical depth buffer, delivered as a non-pipelined state command.			

DWord	Bit	Description
0	31:29	Command Type Default Value: 3h GFXPIPE Format: OpCode
	28:27	Command SubType Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	3D Command Opcode Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	3D Command Sub Opcode Default Value: 0Fh 3DSTATE_HIER_DEPTH_BUFFER Format: OpCode
	15:8	Reserved Project: All Format: MBZ
	7:0	DWord Length Default Value: 1h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:29	Reserved Project: All Format: MBZ
	28:25	Heirarchical Depth Buffer Object Control State Project: [DevSNB] Format: MEMORY_OBJECT_CONTROL_STATE FormatDesc Specifies the memory object control state for the hierarchical depth buffer.
	24:17	Reserved Project: All Format: MBZ

3DSTATE_HIER_DEPTH_BUFFER		
	16:0	**Surface Pitch** Project: All Format: U17 pitch in (Bytes – 1) FormatDesc Range [127, 128K-1] corresponding to [128B, 128KB] also restricted to a multiple of 128B This field specifies the pitch of the hierarchical depth buffer in (#Bytes – 1). **Programming Notes** — Project Since this surface is tiled, the pitch specified must be a multiple of the tile pitch, in the range [128B, 128KB]. — All
2	31:0	**Surface Base Address** Project: All Address: GraphicsAddress[31:0] Surface Type: Hierarchical Depth Buffer This field specifies the starting DWord address of the buffer in mapped Graphics Memory. **Programming Notes** The Hierarchical Depth Buffer can only be mapped to Main Memory (uncached). Since this surface is tiled, the base address must conform to the Per-Surface Tiling Alignment Rules.

7.5.5.4 3DSTATE_CLEAR_PARAMS

7.5.5.4.1 3DSTATE_CLEAR_PARAMS [DevSNB]

3DSTATE_CLEAR_PARAMS packet must follow the DEPTH_BUFFER_STATE packet when HiZ is enabled and the DEPTH_BUFFER_STATE changes.

If HiZ is enabled, you must initialize the clear value by either

a. Perform a depth clear pass to initialize the clear value.

b. Send a 3dstate_clear_params packet with valid = 1

Without one of these events, context switching will fail, as it will try to save off a clear value even though no valid clear value has been set. When context restore happens, HW will restore an uninitialized clear value.

3DSTATE_CLEAR_PARAMS

Project:	DevSNB	**Length Bias:**	2

This command defines the depth clear value, delivered as a non-pipelined state.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 10h 3DSTATE_CLEAR_PARAMS Format: OpCode
	15	**Depth Clear Value Valid** Project: DevSNB Format: Enable FormatDesc This field enables the **Depth Clear Value**. If clear, the depth clear value is obtained from interpolated depth of an arbitrary pixel of the primitive rendered with **Depth Buffer Clear** set in WM_STATE or 3DSTATE_WM. If set, the depth clear value is obtained from the **Depth Clear Value** field of this command.
	14:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2 Project: DevSNB
	7:0	**Reserved**

3DSTATE_CLEAR_PARAMS		
1	31:0	**Depth Clear Value** Project: All Format: IEEE_Float for **Surface Format** of depth buffer: D32_FLOAT_S8X24_UINT: IEEE_Float D32_FLOAT: IEEE_Float D24_UNORM_S8_UINT: U24 UNORM in bits [23:0] D24_UNORM_X8_UINT: U24 UNORM in bits [23:0] D16_UNORM: U16 UNORM in bits [15:0] This field defines the clear value that will be applied to the depth buffer if the **Depth Buffer Clear** field is enabled. It is valid only if **Depth Buffer Clear Value Valid** is set.
2	31:1	**Reserved** Project: All Format: MBZ
	0	**Reserved**

7.6 Barycentric Attribute Interpolation [DevSNB+]

Given hardware clipper and setup, some of the previous flexibility in the algorithm used to interpolate attributes is no longer available. Hardware uses barycentric parameters to aid in attribute interpolation, and these parameters are computed in hardware per-pixel (or per-sample) and delivered in the thread payload to the pixel shader. Also delivered in the payload are a set of vertex deltas (a0, a1, and a2) per channel of each attribute.

There are six different barycentric parameters that can be enabled for delivery in the pixel shader payload. These are enabled via the **Barycentric Interpolation Mode** bits in 3DSTATE_WM.

In the pixel shader kernel, the following computation is done for each attribute channel of each pixel/sample given the corresponding attribute channel a0/a1/a2 and the pixel/sample's b1/b2 barycentric parameters, where A is the value of the attribute channel at that pixel/sample:

A = a0 + (a1 * b1) + (a2 * b2)

7.7 Pixel Shader Thread Generation

After a group of object pixels have been rasterized, the Pixel Shader function is invoked to further compute pixel color/depth information and cause results to be written to rendertargets and/or depth buffers. For each pixel, the Pixel Shader calculates the values of the various vertex attributes that are to be interpolated across the object using the interpolation coefficients. It then executes an API-supplied Pixel Shader Program. Instructions in this program permit the accessing of texture map data, where

Texture Samplers are employed to sample and filter texture maps (see the *Shared Functions* chapter). Arithmetic operations can be performed on the texture data, input pixel information and Pixel Shader Constants in order to compute the resultant pixel color/depth. The Pixel Shader program also allows the pixel to be discarded from further processing. For pixels that are not discarded, the pixel shader must send messages to update one or more render targets with the pixel results.

7.7.1 Pixel Grouping (Dispatch Size) Control

The WM unit can pass a grouping of 2 subspans (8 pixels), 4 subspans (16 pixels) or 8 subspans (32 pixels) to a Pixel Shader thread. Software should take into account the following considerations when determining which groupings to support/enable during operation. This determination involves a tradeoff of these likely conflicting issues. Note that the size of the dispatch has significant impact on the kernel program (it is certainly not transparent to the kernel). Also note that there is no implied spatial relationship between the subspans passed to a PS thread, other than the fact that they come from the same object.

1. **Thread Efficiency**: In general, there is some amount of overhead involved with PS thread dispatch, and if this can be amortized over a larger number of pixels, efficiency will likely increase. This is especially true for very short PS kernels, as may be used for desktop composition, etc.

2. **GRF Consumption**: Processing more pixels per thread will require a larger thread payload and likely more temporary register usage, both of which translate into a requirement for a larger GRF register allocation for the threads. If this increased GRF usage could lead to increased use of scratch space (for spill/fill, etc.) and possibly less efficient use of the EUs (as it would be less likely to find an EU with enough free physical GRF registers to service the thread).

3. **Object Size**: If the number of very small objects (e.g., covering 2 subspans or fewer) is expected to comprise a significant portion of the workload, supporting the 8-pixel dispatch mode may be advantageous. Otherwise there could be a large number of 16-pixel dispatches with only 1 or 2 valid subspans, resulting in low efficiency for those threads.

4. **Intangibles**: Kernel footprint & Instruction Cache impact; Complexity; ….

The groupings of subspans that the WM unit is allowed to include in a PS thread payload is controlled by the **32,16,8 Pixel Dispatch Enable** state variables programmed in WM_STATE. Using these state variables, the WM unit will attempt to dispatch the largest allowed grouping of subspans. The following table lists the possible combinations of these state variables.

Note: in the table below, the Valid column indicates which products that combination is supported on. Combinations of dispatch enables not listed in the table are not available on any product.

A: Valid on all products

B: Not valid on [DevSNB] if 4x PERPIXEL mode with pixel shader computed depth.

D: Valid on all products, except when in non-1x PERSAMPLE mode (applies to [DevSNB+] only). Not valid on [DevSNB] if 4x PERPIXEL mode with pixel shader computed depth.

E: Not valid on [DevSNB] if 4x PERPIXEL mode with pixel shader computed depth.

F: Valid on all products, except not valid on [DevSNB] if 4x PERPIXEL mode with pixel shader computed depth.

For [DevSNB**],** there is only one kernel start pointer (KSP) specified in WM_STATE, with other kernels being entered via an offset from the single KSP as follows:

SP[0] = KSP

KSP[1] = KSP+1

KSP[2] = KSP+2

KSP[3] = KSP+3

For **[DevSNB]**, each of the three KSP values is separately specified. In addition, each kernel has a separately-specified GRF register count.

Table 22. Variable Pixel Dispatch

Contiguous 64 Pixel Dispatch Enable	Contiguous 32 Pixel Dispatch Enable	32 Pixel Dispatch Enable	16 Pixel Dispatch Enable	8 Pixel Dispatch Enable	Valid	IP for n-pixel Contiguous Dispatch		IP for n-pixel Dispatch (KSP offsets are in 128-bit instruction units)		
						n=64	n=32	n=32	n=16	n=8
0	0	0	0	1	A					KSP [0]
0	0	0	1	0	F				KSP [0]	
0	0	0	1	1	D				KSP [2]	KSP [0]
0	0	1	0	0	B			KSP [0]		
0	0	1	1	0	E			KSP [1]	KSP [2]	
0	0	1	1	1	D			KSP [1]	KSP [2]	KSP [0]
0	1	0	0	0	C		KSP [0]			
0	1	1	0	0	C		KSP [1]	KSP [0]		
0	1	1	1	0	C		KSP [2]	KSP [1]	KSP [0]	
1	0	0	0	0	C	KSP [0]				
1	0	1	0	0	C	KSP [1]		KSP [0]		
1	0	1	1	0	C	KSP [2]		KSP [1]	KSP [0]	
1	1	0	0	0	C	KSP [1]	KSP [0]			
1	1	1	0	0	C	KSP [2]	KSP [1]	KSP [0]		

7.7.2 Multisampling Effects on Pixel Shader Dispatch [DevSNB+]

The pixel shader payloads are defined in terms of subspans and pixels. The slots in the pixel shader thread previously mapped 1:1 with pixels. With multisampling, a slot could contain a pixel or may just contain a single sample, depending on the mode. Payload definitions now refer to "slot" to make the definition independent of multisampling mode.

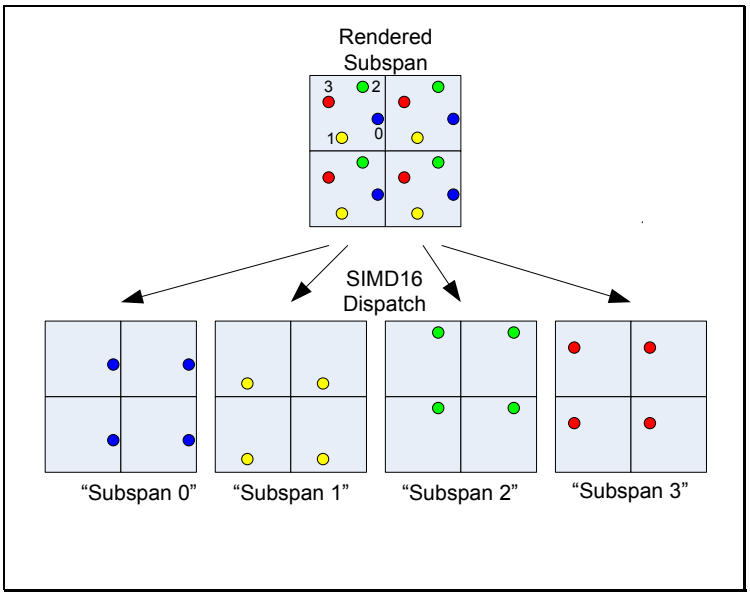
7.7.2.1 MSDISPMODE_PERPIXEL Thread Dispatch

In PERPIXEL mode, the pixel shader kernel still works on 2/4/8 separate subspans, depending on dispatch mode. The fact that rasterization and the depth/stencil tests are being performed on a per-sample (not per-pixel) basis is transparent to the pixel shader kernel.

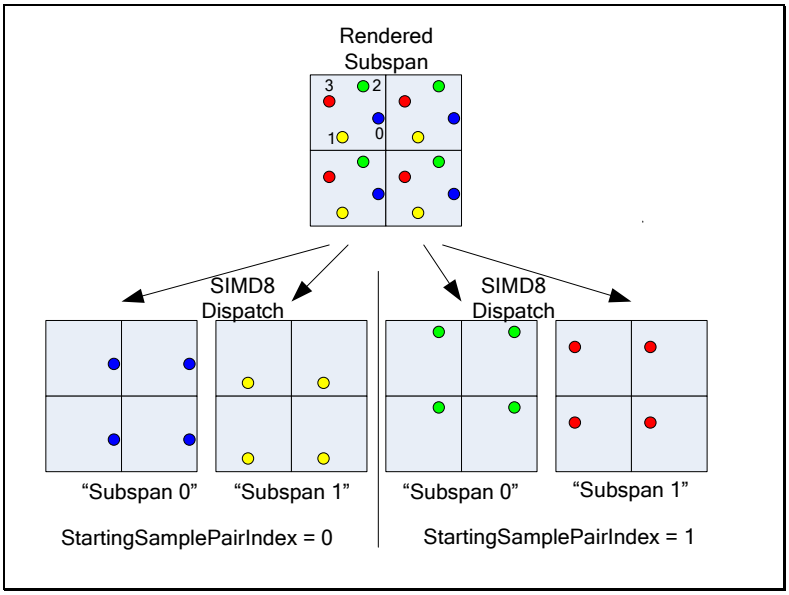
7.7.2.2 MSDISPMODE_PERSAMPLE Thread Dispatch

In PERSAMPLE mode, the pixel shader needs to operate on a sample vs. pixel basis (although this collapses in NUMSAMPLES_1 mode). Instead of processing strictly different subspans in parallel , the PS kernel processes different sample indices of one or more subspans in parallel. For example, a SIMD16 dispatch in PERSAMPLE/NUMSAMPLES_4 mode would operate on a single subspan, with the usual "4 Subspan0 pixel slots" used for the "4 Sample0 locations of the (single) subspan". Subspan1 slots would be used for the Sample1 locations, and so on. This layout allows the pixel shader to compute derivatives/LOD based on deltas between corresponding sample locations in the subspan in the same fashion as LEGACY pixel shader execution.

Depending on the dispatch mode (8/16/32 pixels) and multisampling mode (1X/4X), there are different mappings of subspans/samples onto dispatches and slots-within-dispatch. In some cases, more than one subspan may be included in a dispatch, while in other cases multiple dispatches are be required to process all samples for a single subspan. In the latter case, the **StartingSamplePairIndex** value is included in the payload header so the Render Target Write message will access the correct samples with each message.



PERSAMPLE SIMD16 4X Dispatch



PERSAMPLE SIMD8 4X Dispatch

The following table provides the complete dispatch/slot mappings for all the MS/Dispatch combinations.

Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
SIMD32	1X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[1].Pixel[3:0].Sample[0] Slot[11:8] = Subspan[2].Pixel[3:0].Sample[0] Slot[15:12] = Subspan[3].Pixel[3:0].Sample[0] Slot[19:16] = Subspan[4].Pixel[3:0].Sample[0] Slot[23:20] = Subspan[5].Pixel[3:0].Sample[0] Slot[27:24] = Subspan[6].Pixel[3:0].Sample[0] Slot[31:28] = Subspan[7].Pixel[3:0].Sample[0]
	4X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1] Slot[11:8] = Subspan[0].Pixel[3:0].Sample[2] Slot[15:12] = Subspan[0].Pixel[3:0].Sample[3]

Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
		Slot[19:16] = Subspan[1].Pixel[3:0].Sample[0] Slot[23:20] = Subspan[1].Pixel[3:0].Sample[1] Slot[27:24] = Subspan[1].Pixel[3:0].Sample[2] Slot[31:28] = Subspan[1].Pixel[3:0].Sample[3]
	8X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1] Slot[11:8] = Subspan[0].Pixel[3:0].Sample[2] Slot[15:12] = Subspan[0].Pixel[3:0].Sample[3] Slot[19:16] = Subspan[0].Pixel[3:0].Sample[4] Slot[23:20] = Subspan[0].Pixel[3:0].Sample[5] Slot[27:24] = Subspan[0].Pixel[3:0].Sample[6] Slot[31:28] = Subspan[0].Pixel[3:0].Sample[7]
SIMD16	1X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[1].Pixel[3:0].Sample[0] Slot[11:8] = Subspan[2].Pixel[3:0].Sample[0] Slot[15:12] = Subspan[3].Pixel[3:0].Sample[0]
	4X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1] Slot[11:8] = Subspan[0].Pixel[3:0].Sample[2] Slot[15:12] = Subspan[0].Pixel[3:0].Sample[3]
	8X	Dispatch[i]: (i=0, 2) SSPI = i Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1] Slot[11:8] = Subspan[0].Pixel[3:0].Sample[SSPI*2+2] Slot[15:12] = Subspan[0].Pixel[3:0].Sample[SSPI*2+3]

Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
SIMD8	1X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[1].Pixel[3:0].Sample[0]
	4X	Dispatch[i]: (i=0..1) SSPI = i Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1]
	8X	Dispatch[i]: (i=0, 1, 2, 3) SSPI = i Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1]

7.7.3 PS Thread Payload for Normal Dispatch

The following tables list all possible contents included in a PS thread payload, in the order they are provided. Certain portions of the payload are optional, in which case the corresponding phase is skipped.

This payload does not apply to the contiguous dispatch modes on **[DevCTG+]**. The payload for these modes are documented in the section titled *PS Thread Payload for Contiguous Dispatch*.

7.7.3.1 PS Thread Payload for Normal Dispatch [DevSNB+]

The following payload applies to **[DevSNB]**. All registers are numbered starting at 0, but many registers are skipped depending on configuration. This causes all registers below to be renumbered to fill in the skipped locations. The only case where actual registers may be skipped is immediately before the constant data and again before the setup data.

DWord	Bit	Description
R0.7	31	Snapshot Flag
	30:24	Reserved
	23:0	**Primitive Thread ID:** This field contains the primitive thread count passed to the Windower from the Strips Fans Unit. Format: Reserved for HW Implementation Use.

DWord	Bit	Description
R0.6	31:24	Reserved
	23:0	**Thread ID:** This field contains the thread count which is incremented by the Windower for every thread that is dispatched. Format: Reserved for HW Implementation Use.
R0.5	31:10	**Scratch Space Pointer:** Specifies the 1K-byte aligned pointer to the scratch space available for this PS thread. This is specified as an offset to the **General State Base Address**. Format = GeneralStateOffset[31:10]
	9:8	**Color Code:** This ID is assigned by the Windower unit and is used to track synchronizng events. Format: Reserved for HW Implementation Use.
	7:0	**FFTID:** This ID is assigned by the WM unit and is a identifier for the thread. It is used to free up resources used by the thread upon thread completion. Format: Reserved for HW Implementation Use.
R0.4	31:5	**Binding Table Pointer:** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved
R0.3	31:5	**Sampler State Pointer:** Specifies the 32-byte aligned pointer to the Sampler State table. It is specified as an offset from the **Dynamic State Base Address**. Format = DynamicStateOffset[31:5]
	4	Reserved
	3:0	**Per Thread Scratch Space:** Specifies the amount of scratch space allowed to be used by this thread. Programming Notes: This amount is available to the kernel for information only. It will be passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port will ignore it. Format = U4 Range = [0,11] indicating [1k bytes, 2M bytes] in powers of two
R0.2	31:0	Reserved : delivered as zeros (reserved for message header fields)

DWord	Bit	Description
R0.1	31:6	**Color Calculator State Pointer:**Specifies the 64-byte aligned pointer to the Color Calculator state (COLOR_CALC_STATE structure in memory). It is specified as an offset from the **Dynamic State Base Address**. This value is eventually passed to the ColorCalc function in the DataPort and is used to fetch the corresponding CC_STATE data. Format = DynamicStateOffset[31:6]
	5:0	Reserved
R0.0	31	Reserved
	30:27	**Viewport Index.** Specifies the index of the viewport currently being used. Format = U4 Range = [0,15]
	26:16	**Render Target Array Index:**Specifies the array index to be used for the following surface types: SURFTYPE_1D: specifies the array index. Range = [0,511] SURFTYPE_2D: specifies the array index. Range = [0,511] SURFTYPE_3D: specifies the “r” coordinate. Range = [0,2047] SURFTYPE_CUBE: specifies the face identifier. Range = [0,5] face Render Target Array Index +x 0 -x 1 +y 2 -y 3 +z 4 -z 5 Format = U11
	15	**Front/Back Facing Polygon:** Determines whether the polygon is front or back facing. Used by the render cache to determine which stencil test state to use. 0: Front Facing 1: Back Facing
	14	Reserved

DWord	Bit	Description
	13	**Source Depth to Render Target:** Indicates that source depth will be sent to the render target
	12	**oMask to Render Target:** Indicates that oMask will be sent to the render target
	11:9	Reserved
	8:7	Reserved for expansion of **Starting Sample Pair Index**
	6	**Starting Sample Pair Index:** indicates the index of the first sample pair of the dispatch Format = U1 [**DevSNB+**]: Range = [0,1]
	5	Reserved
	4:0	**Primitive Topology Type:** This field identifies the Primitive Topology Type associated with the primitive spawning this object. The WM unit does not modify this value (e.g., objects within POINTLIST topologies see POINTLIST). Format: (See 3DPRIMITIVE command in *3D Pipeline*)
R1.7	31:16	**Pixel/Sample Mask (SubSpan[3:0]) :** Indicates which pixels within the four subspans are lit. If 32 pixel dispatch is enabled, this field contains the pixel mask for the first four subspans. **Note:** This is not a duplicate of the Dispatch Mask that is delivered to the thread. The dispatch mask has all pixels within a subspan as active if any of them are lit to enable LOD calculations to occur correctly. This field must not be modified by the Pixel Shader kernel.
	15:0	**Pixel/Sample Mask Copy (SubSpan[3:0]) :** This is a duplicate copy of the pixel mask. This copy can be modified as the pixel shader thread executes in order to turn off pixels based on kill instructions.
R1.6	31:0	Reserved
R1.5	31:16	**Y3:** Y coordinate (screen space) for upper-left pixel of subspan 3 (slot 12) Format = U16
	15:0	**X3:** X coordinate (screen space) for upper-left pixel of subspan 3 (slot 12) Format = U16
R1.4	31:16	**Y2 :** Y coordinate (screen space) for upper-left pixel of subspan 2 (slot 8) Format = U16

DWord	Bit	Description
	15:0	**X2 :** X coordinate (screen space) for upper-left pixel of subspan 2 (slot 8) Format = U16
R1.3	31:16	**Y1 :** Y coordinate (screen space) for upper-left pixel of subspan 1 (slot 4) Format = U16
	15:0	**X1 :** X coordinate (screen space) for upper-left pixel of subspan 1 (slot 4) Format = U16
R1.2	31:16	**Y0 :** Y coordinate (screen space) for upper-left pixel of subspan 0 (slot 0) Format = U16
	15:0	**X0 :** X coordinate (screen space) for upper-left pixel of subspan 0 (slot 0) Format = U16
R1.1	31:0	Reserved
R1.0	31:0	Reserved
		R2: delivered only if this is a *32-pixel dispatch*.
R2.7	31:16	**Pixel/Sample Mask (SubSpan[7:4]) :** Indicates which pixels within the upper four subspans are lit. This field is valid only when the 32 pixel dispatch state is enabled. This field must not be modified by the pixel shader thread. Note: This is not a duplicate of the dispatch mask that is delivered to the thread. The dispatch mask has all pixels within a subspan as active if any of them are lit to enable LOD calculations to occur correctly. This field must not be modified by the Pixel Shader kernel.
	15:0	**Pixel/Sample Mask Copy (SubSpan[7:4]) :** This is a duplicate copy of pixel mask for the upper 16 pixels. This copy will be modified as the pixel shader thread executes to turn off pixels based on kill instructions.
R2.6	31:0	Reserved
R2.5	31:16	**Y7:** Y coordinate (screen space) for upper-left pixel of subspan 7 (slot 28) Format = U16
	15:0	**X7:** X coordinate (screen space) for upper-left pixel of subspan 7 (slot 28) Format = U16
R2.4	31:16	Y6

DWord	Bit	Description
	15:0	X6
R2.3	31:16	Y5
	15:0	X5
R2.2	31:16	Y4
	15:0	X4
R2.1	31:0	Reserved
R2.0	31:0	Reserved
		R3-R26: delivered only if the corresponding **Barycentric Interpolation Mode** (WM_STATE) bit is set. Register phases containing Slot 8-15 data are not delivered in *8-pixel dispatch* mode.
R3.7	31:0	Perspective Pixel Location Barycentric[1] for Slot 7 This and the next register phase is only included if the corresponding enable bit in **Barycentric Interpolation Mode** is set. Format = IEEE_Float
R3.6	31:0	Perspective Pixel Location Barycentric[1] for Slot 6
R3.5	31:0	Perspective Pixel Location Barycentric[1] for Slot 5
R3.4	31:0	Perspective Pixel Location Barycentric[1] for Slot 4
R3.3	31:0	Perspective Pixel Location Barycentric[1] for Slot 3
R3.2	31:0	Perspective Pixel Location Barycentric[1] for Slot 2
R3.1	31:0	Perspective Pixel Location Barycentric[1] for Slot 1
R3.0	31:0	Perspective Pixel Location Barycentric[1] for Slot 0
R4		Perspective Pixel Location Barycentric[2] for Slots 7:0
R5.7	31:0	Perspective Pixel Location Barycentric[1] for Slot 15
R5.6	31:0	Perspective Pixel Location Barycentric[1] for Slot 14
R5.5	31:0	Perspective Pixel Location Barycentric[1] for Slot 13
R5.4	31:0	Perspective Pixel Location Barycentric[1] for Slot 12

DWord	Bit	Description
R5.3	31:0	Perspective Pixel Location Barycentric[1] for Slot 11
R5.2	31:0	Perspective Pixel Location Barycentric[1] for Slot 10
R5.1	31:0	Perspective Pixel Location Barycentric[1] for Slot 9
R5.0	31:0	Perspective Pixel Location Barycentric[1] for Slot 8
R6		Perspective Pixel Location Barycentric[2] for Slots 15:8
R7:10		Perspective Centroid Barycentric
R11:14		Perspective Sample Barycentric
R15:18		Linear Pixel Location Barycentric
R19:22		Linear Centroid Barycentric
R23:26		Linear Sample Barycentric
		R27: delivered only if Pixel Shader Uses Source Depth is set.
R27.7	31:0	Interpolated Depth for Slot 7 Format = IEEE_Float This and the next register phase is only included if **Pixel Shader Uses Source Depth** (WM_STATE) is set.
R27.6	31:0	Interpolated Depth for Slot 6
R27.5	31:0	Interpolated Depth for Slot 5
R27.4	31:0	Interpolated Depth for Slot 4
R27.3	31:0	Interpolated Depth for Slot 3
R27.2	31:0	Interpolated Depth for Slot 2
R27.1	31:0	Interpolated Depth for Slot 1
R27.0	31:0	Interpolated Depth for Slot 0
		R28: delivered only if **Pixel Shader Uses Source Depth** is set and this is not an *8-pixel dispatch*.
R28.7	31:0	Interpolated Depth for Slot 15
R28.6	31:0	Interpolated Depth for Slot 14

DWord	Bit	Description
R28.5	31:0	Interpolated Depth for Slot 13
R28.4	31:0	Interpolated Depth for Slot 12
R28.3	31:0	Interpolated Depth for Slot 11
R28.2	31:0	Interpolated Depth for Slot 10
R28.1	31:0	Interpolated Depth for Slot 9
R28.0	31:0	Interpolated Depth for Slot 8
		R29: delivered only if Pixel Shader Uses Source W is set.
R29.7	31:0	Interpolated W for Slot 7 Format = IEEE_Float This and the next register phase is only included if **Pixel Shader Uses Source W** (WM_STATE) is set
R29.6	31:0	Interpolated W for Slot 6
R29.5	31:0	Interpolated W for Slot 5
R29.4	31:0	Interpolated W for Slot 4
R29.3	31:0	Interpolated W for Slot 3
R29.2	31:0	Interpolated W for Slot 2
R29.1	31:0	Interpolated W for Slot 1
R29.0	31:0	Interpolated W for Slot 0
		R30: delivered only if **Pixel Shader Uses Source W** is set and this is not an *8-pixel dispatch*.
R30.7	31:0	Interpolated W for Slot 15
R30.6	31:0	Interpolated W for Slot 14
R30.5	31:0	Interpolated W for Slot 13
R30.4	31:0	Interpolated W for Slot 12
R30.3	31:0	Interpolated W for Slot 11
R30.2	31:0	Interpolated W for Slot 10

DWord	Bit	Description
R30.1	31:0	Interpolated W for Slot 9
R30.0	31:0	Interpolated W for Slot 8
		R31: delivered only if **Position XY Offset Select** is either POSOFFSET_CENTROID or POSOFFSET_SAMPLE
R31.7	31:24	Position Offset Y for Slot 15 This field contains either the CENTROID or SAMPLE position offset for Y, depending on the state of **Position XY Offset Select**. Format = U4.4 Range = [0.0,1.0)
	23:16	Position Offset X for Slot 15 This field contains either the CENTROID or SAMPLE position offset for X, depending on the state of **Position XY Offset Select**. Format = U4.4 Range = [0.0,1.0)
	15:8	Position Offset Y for Slot 14
	7:0	Position Offset X for Slot 14
R31.6	31:24	Position Offset Y for Slot 13
	23:16	Position Offset X for Slot 13
	15:8	Position Offset Y for Slot 12
	7:0	Position Offset X for Slot 12
R31.5:4		Position Offset X/Y for Slot[11:8]
R31.3:2		Position Offset X/Y for Slot[7:4]
R31.1:0		Position Offset X/Y for Slot[3:0]
		R32-R55: delivered only if the corresponding **Barycentric Interpolation Mode** (WM_STATE) bit is set and this is a *32-pixel dispatch*.

DWord	Bit	Description
R32.7	31:0	Perspective Pixel Location Barycentric[1] for Slot 23 This and the next register phase is only included if the corresponding enable bit in **Barycentric Interpolation Mode** is set. Format = IEEE_Float
R32.6	31:0	Perspective Pixel Location Barycentric[1] for Slot 22
R32.5	31:0	Perspective Pixel Location Barycentric[1] for Slot 21
R32.4	31:0	Perspective Pixel Location Barycentric[1] for Slot 20
R32.3	31:0	Perspective Pixel Location Barycentric[1] for Slot 19
R32.2	31:0	Perspective Pixel Location Barycentric[1] for Slot 18
R32.1	31:0	Perspective Pixel Location Barycentric[1] for Slot 17
R32.0	31:0	Perspective Pixel Location Barycentric[1] for Slot 16
R33		Perspective Pixel Location Barycentric[2] for Slots 23:16
R34.7	31:0	Perspective Pixel Location Barycentric[1] for Slot 31
R34.6	31:0	Perspective Pixel Location Barycentric[1] for Slot 30
R34.5	31:0	Perspective Pixel Location Barycentric[1] for Slot 29
R34.4	31:0	Perspective Pixel Location Barycentric[1] for Slot 28
R34.3	31:0	Perspective Pixel Location Barycentric[1] for Slot 27
R34.2	31:0	Perspective Pixel Location Barycentric[1] for Slot 26
R34.1	31:0	Perspective Pixel Location Barycentric[1] for Slot 25
R34.0	31:0	Perspective Pixel Location Barycentric[1] for Slot 24
R35		Perspective Pixel Location Barycentric[2] for Slots 31:24
R36:39		Perspective Centroid Barycentric
R40:43		Perspective Sample Barycentric
R44:47		Linear Pixel Location Barycentric
R48:51		Linear Centroid Barycentric

DWord	Bit	Description
R52:55		Linear Sample Barycentric
		R56-R57: delivered only if **Pixel Shader Uses Source Depth** is set and this is a *32-pixel dispatch*.
R56.7	31:0	Interpolated Depth for Slot 23 Format = IEEE_Float This and the next register phase is only included if **Pixel Shader Uses Source Depth** (WM_STATE) bit is set.
R56.6	31:0	Interpolated Depth for Slot 22
R56.5	31:0	Interpolated Depth for Slot 21
R56.4	31:0	Interpolated Depth for Slot 20
R56.3	31:0	Interpolated Depth for Slot 19
R56.2	31:0	Interpolated Depth for Slot 18
R56.1	31:0	Interpolated Depth for Slot 17
R56.0	31:0	Interpolated Depth for Slot 16
R57.7	31:0	Interpolated Depth for Slot 31
R57.6	31:0	Interpolated Depth for Slot 30
R57.5	31:0	Interpolated Depth for Slot 29
R57.4	31:0	Interpolated Depth for Slot 28
R57.3	31:0	Interpolated Depth for Slot 27
R57.2	31:0	Interpolated Depth for Slot 26
R57.1	31:0	Interpolated Depth for Slot 25
R57.0	31:0	Interpolated Depth for Slot 24
		R58-R59: delivered only if **Pixel Shader Uses Source W** is set and this is a *32-pixel dispatch*.

DWord	Bit	Description
R58.7	31:0	Interpolated W for Slot 23 Format = IEEE_Float This and the next register phase is only included if **Pixel Shader Uses Source W** (WM_STATE) bit is set.
R58.6	31:0	Interpolated W for Slot 22
R58.5	31:0	Interpolated W for Slot 21
R58.4	31:0	Interpolated W for Slot 20
R58.3	31:0	Interpolated W for Slot 19
R58.2	31:0	Interpolated W for Slot 18
R58.1	31:0	Interpolated W for Slot 17
R58.0	31:0	Interpolated W for Slot 16
R59.7	31:0	Interpolated W for Slot 31
R59.6	31:0	Interpolated W for Slot 30
R59.5	31:0	Interpolated W for Slot 29
R59.4	31:0	Interpolated W for Slot 28
R59.3	31:0	Interpolated W for Slot 27
R59.2	31:0	Interpolated W for Slot 26
R59.1	31:0	Interpolated W for Slot 25
R59.0	31:0	Interpolated W for Slot 24
		R60: delivered only if **Position XY Offset Select** is either POSOFFSET_CENTROID or POSOFFSET_SAMPLE and this is a *32-pixel dispatch*.
R60.7	31:24	Position Offset Y for Slot 31 This field contains either the CENTROID or SAMPLE position offset for Y, depending on the state of **Position XY Offset Select**. Format = U4.4 Range = [0.0,1.0)

DWord	Bit	Description
	23:16	Position Offset X for Slot 31 This field contains either the CENTROID or SAMPLE position offset for X, depending on the state of **Position XY Offset Select**. Format = U4.4 Range = [0.0,1.0)
	15:8	Position Offset Y for Slot 30
	7:0	Position Offset X for Slot 30
R60.6	31:24	Position Offset Y for Slot 29
	23:16	Position Offset X for Slot 29
	15:8	Position Offset Y for Slot 28
	7:0	Position Offset X for Slot 28
R60.5:4		Position Offset X/Y for Slot[27:24]
R60.3:2		Position Offset X/Y for Slot[23:20]
R60.1:0		Position Offset X/Y for Slot[19:16]
		Optional Padding before the Start of Constant/Setup Data The locations between the end of the Optional Payload Header and the location programmed via **Dispatch GRF Start Register for Constant/Setup Data** are considered “padding” and Reserved. (see below)
optional, multiple of 8 DWs	31:0	Reserved
		The **Dispatch GRF Start Register for Constant/Setup Data** state variable in 3DSTATE_WM is used to define the starting location of the constant and setup data within the PS thread payload. This control is provided to allow this data to be located at a fixed location within thread payloads, regardless of the amount of data in the Optional Payload Header. This permits the kernel to use direct GRF addressing to access the constant/setup data, regardless of the optional parameters being passed (as these are determined on-the-fly by the WM unit).

DWord	Bit	Description
		Constant Data (optional) : Some amount of constant data (possible none) can be extracted from the push constant buffer (PCB) and passed to the thread following the R0 Header. The amount of data provided is defined by the sum of the read lengths in the last 3DSTATE_CONSTANT_PS command (taking the buffer enables into account). The Constant Data arrives in a non-interleaved format.
optional, multiple of 8 DWs	31:0	Constant Data
		Setup Data (Attribute Vertex Deltas) Output data from the SF stage is delivered in the PS thread payload. The amount of data is determined by the **Number of Output Attributes** field in 3DSTATE_SF. Each register contains two channels of one attribute. Thus, the total number of registers sent is equal to 2 * Number of Output Attributes.
Rp.7	31:0	**a0[0].y** – a0 vertex delta for Attribute0.y Format = IEEE_Float
Rp.6	31:0	Reserved
Rp.5	31:0	**a2[0].y** – a2 vertex delta for Attribute0.y Format = IEEE_Float
Rp.4	31:0	**a1[0].y** – a1 vertex delta for Attribute0.y Format = IEEE_Float
Rp.3	31:0	**a0[0].x** – a0 vertex delta for Attribute0.x
Rp.2	31:0	Reserved
Rp.1	31:0	**a2[0].x** – a2 vertex delta for Attribute0.x
Rp.0	31:0	**a1[0].x** – a1 vertex delta for Attribute0.x
R(p+1).7	31:0	**a0[0].w** – a0 vertex delta for Attribute0.w
R(p+1).6	31:0	Reserved
R(p+1).5	31:0	**a2[0].w** – a2 vertex delta for Attribute0.w
R(p+1).4	31:0	**a1[0].w** – a1 vertex delta for Attribute0.w
R(p+1).3	31:0	**a0[0].z** – a0 vertex delta for Attribute0.z

DWord	Bit	Description
R(p+1).2	31:0	Reserved
R(p+1).1	31:0	**a2[0].z** – a2 vertex delta for Attribute0.z
R(p+1).0	31:0	**a1[0].z** – a1 vertex delta for Attribute0.z
R(p+2):Rq		Vertex deltas for additional attributes in numerical order See definition of Rp and R(p+1) for formats.

7.8 Other WM Functions

7.8.1 Statistics Gathering

If **Statistics Enable** is set in WM_STATE or 3DSTATE_WM, the Windower increments the PS_INVOCATIONS_COUNT register once for each unmasked pixel (or sample) that is *dispatched* to a Pixel Shader thread. If **Early Depth Test Enable** is set it is possible for pixels or samples to be discarded prior to reaching the Pixel Shader due to failing the depth or stencil test. PS_INVOCATIONS_COUNT will still be incremented for these pixels or samples since the depth test occurs after the pixel shader from the point of view of SW.

8. Color Calculator (Output Merger)

Note: The Color Calculator logic resides in the Render Cache backing Data Port (DAP) shared function. It is described in this chapter as the Color Calc functions are naturally an extension of the 3D pipeline past the WM stage. See the DataPort chapter for details on the messages used by the Pixel Shader to invoke Color Calculator functionality.

The *Color Calculator* function within the Data Port shared function completes the processing of rasterized pixels after the pixel color and depth have been computed by the Pixel Shader. This processing is initiated when the pixel shader thread sends a Render Target Write message (see *Shared Functions*) to the Render Cache. (Note that a single pixel shader thread may send multiple Render Target Write messages, with the result that multiple render targets get updated). The pixel variables pass through a pipeline of fixed (yet programmable) functions, and the results are conditionally written into the appropriate buffers.

[DevSNB+]: The word "pixel" used in this section is effectively replaced with the word "sample" if multisample rasterization is enabled.

Pipeline Stage	Description
Alpha Coverage [DevSNB+]	[DevSNB+]It generates coverage masks using AlphaToCoverage AND/OR AlphaToOne functions based on src0.alpha.
Alpha Test	Compare pixel alpha with reference alpha and conditionally discard pixel
Stencil Test	Compare pixel stencil value with reference and forward result to Buffer Update stage
Depth Test	Compare pix.Z with corresponding Z value in the Depth Buffer and forward result to Buffer Update stage
Color Blending	Combine pixel color with corresponding color in color buffer according to programmable function
Gamma Correction	Adjust pixel's color according to gamma function for SRGB destination surfaces.
Color Quantization	Convert "full precision" pixel color values to fixed precision of the color buffer format
Logic Ops	Combine pixel color logically with existing color buffer color (mutually exclusive with Color Blending)
Buffer Update	Write final pixel values to color and depth buffers or discard pixel without update

The following logic describes the high-level operation of the Pixel Processing pipeline:

```
PixelProcessing() {
   AlphaCoverage()// [DevSNB+]
   AlphaTest()
   DepthBufferCoordinateOffsetDisable
   StencilTest()
   DepthTest()
   ColorBufferBlending()
   GammaCorrection()
   ColorQuantization()
   LogicalOps()
   BufferUpdate()
}
```

[SNB WA]: Errata BK83/BJ84 - Display Corruption may be seen after graphics voltage rail (VCC_AXG) power up from 0V

This workaround must be applied after any power-up condition when graphics voltage is starting from zero volts (examples: initial power up, S3/S4 resume, etc..).

In order to set the initial state of the 3D engine (Color Calculator) to avoid incorrect 3D rendering after such a power up cycle, driver software must implement the following before any other 3D rendering occurs:

Set a render target with a size of at least 1 page, set base graphics address of 64K (0x10000)

- Render target should be set for SURFACE_TYPE_NULL.

Set a depth (Z) buffer with a size of at least 1 page, set base graphics address of 128K (0x20000)

- Depth Test should be set to ENABLE
- Depth Test Function should be COMPAREFUNCTION_ALWAYS
- Depth Write DISABLE
- HiZ should be DISABLE

Generic Surface Settings for both Buffers

- Surface X,Y offsets should be ZERO
- LOD should be set to ZERO

Render array target index must be set to ZERO

Page table entry for the addresses should be valid (do not generate a page fault)

Do not use Stencil Buffer

Render a single point list primitive with these buffers

Specify X=0, Y=0 for render location

Pixel Sample Point Value set to UPPER LEFT

No textures or sampling required.

Per state settings, the pixel must be rendered (do not clip, cull, or scissor)

With these settings, the pipeline will be cleared and the color calculator will properly render subsequent 3D primitives. With the pipeline state specified above (SURFACE_TYPE_NULL and Depth Write DISABLE), the workaround will not write any rendered pixel data to memory.

8.1.1 Alpha Coverage [DevSNB+]

Alpha coverage logic is supported for DevSNB+ and can be controlled using three state variables:

- **AlphaToCoverage Enable**, when enabled Color Calculator modifies the sample mask. This function (along with AlphaToOne) come at the top of the pixel pipeline. The sample's Source0.Alpha value (possibly being replicated from the pixel's Source0.Alpha) is used to compute a (optionally dithered) 1/2/4-bit mask (depending on NumSamples).
- The **AlphaToCoverage Dither Enable** SV is used to control the dithering of the AlphaToCoverage mask. The bit corresponding to the sample# is then ANDed with the sample's incoming mask bits – allowing the sample to be masked off depending on alpha.
- **AlphaToOne Enable,** when enabled, Color Calculator must replace Source0.Alpha (if present) with 1.0f.
- If AlphaToCoverage is disabled, AlphaToCoverage Dither does not have any impact.

NOTE:

- Src0.alpha needs to be first multiplied with AA alpha before applying AlphaToCoverage and AlphaToOne functions.
- An alpha value of NaN results in a no coverage (zero) mask.
- **[DevSNB]:** When NumSamples = 1, AlphaToCoverage and AlphaTo Coverage Dither both must be disabled.
- Alpha values from the pixel shader are treated as FLOAT32 format for computing the AlphaToCoverage Mask.

8.1.2 Alpha Test

The Alpha Test function can be used to discard pixels based on a comparison between the incoming pixel's alpha value and the **Alpha Test Reference** state variable in COLOR_CALC_STATE. This operation can be used to remove transparent or nearly-transparent pixels, though other uses for the alpha channel and alpha test are certainly possible.

This function is enabled by the **Alpha Test Enable** state variable in COLOR_CALC_STATE. If ENABLED, this function compares the incoming pixel's alpha value (*pixColor.Alpha*) and the reference alpha value specified by via the **Alpha Test Reference** state variable in COLOR_CALC_STATE. The comparison performed is specified by the **Alpha Test Function** state variable in COLOR_CALC_STATE.

The **Alpha Test Format** state variable is used to specify whether Alpha Test is performed using fixed-point (UNORM8) or FLOAT32 values. Accordingly, it determines whether the **Alpha Reference Value** is passed in a UNORM8 or FLOAT32 format. If UNORM8 is selected, the pixel's alpha value will be converted from floating-point to UNORM8 before the comparison.

Pixels that pass the Alpha Test proceed for further processing. Those that fail are discarded at this point in the pipeline.

If **Alpha Test Enable** is DISABLED, this pipeline stage has no effect.

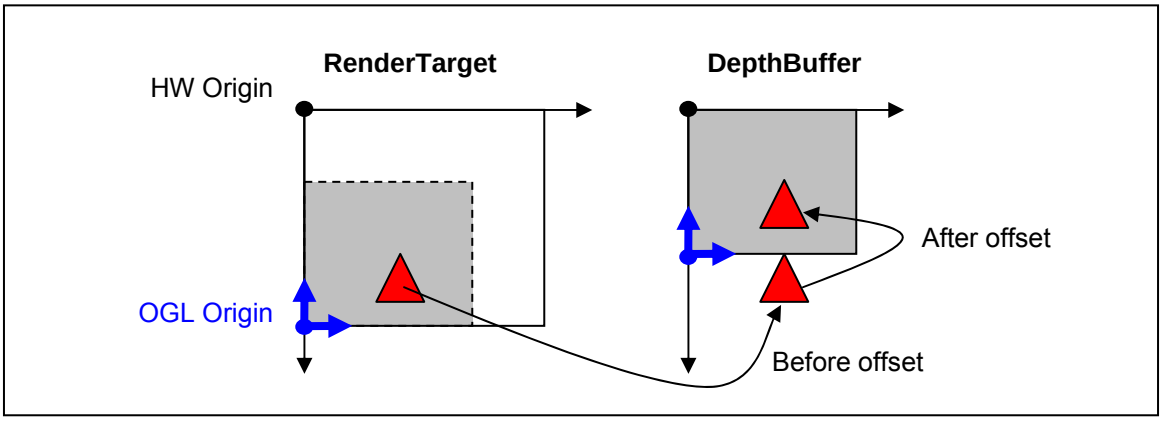
[DevSNB+]: The Alpha Test function is supported in conjunction with Multiple Render Targets (MRTs). If delivered in the incoming render target write message, source 0 alpha is used to perform the alpha test. If source 0 alpha is not delivered, the normal alpha value is used to perform the alpha test.

8.1.3 Depth Coordinate Offset

The Depth Coordinate Offset function applies a programmable constant offset to the RenderTarget X,Y screen space coordinates in order to generate DepthBuffer coordinates.

The function has been specifically added to allow the OpenGL driver to deal with a RenderTarget and DepthBuffer of differing sizes. This condition isn't an issue for the D3D driver, as D3D defines a upper-left screen coordinate origin which matches the HW rasterizer – as long as the application limits rendering to the smaller of the RT/DepthBuffer extents, no special logic is required.

In contrast, OpenGL defines a lower-left screen coordinate origin. This requires the driver to incorporate a "Y coordinate flipping" transformation into the viewport mapping function. The Y extent of the RT is used in this flipping transformation. If the DepthBuffer extent is different, the wrong pixel Y locations within the DepthBuffer will be accessed.



The least expensive solution is to provide a translation offset to be applied to the post-viewport-mapped DepthBuffer Y pixel coordinate, effectively allowing the alignment of the lower-left origins of the RT and DepthBuffer. [Note that the previous DBCOD feature performed an optional translation of post-viewport-mapping RT pixel (screen) coordinates to generate DepthBuffer pixel (window) coordinates. Specifically, the Draw Rect Origin X,Y state could be subtracted from the RT pixel coordinates.]

This function uses **Depth Coordinate Offset X,Y** state (signed 16-bit values in 3DSTATE_DEPTH_RECTANGLE) that is unconditionally added to the RT pixel coordinates to generate DepthBuffer pixel coordinates.

The previous DBCOB feature can be supported by having the driver program Depth Coordinate X,Y Offset to the two's complement of the the Draw Rect Origin. By programming Depth Coordinate X,Y Offset to zeros, the current "normal" operation (DBCOD disabled) can be achieved.

Programming Restrictions:

- Only simple 2D RTs are supported (no mipmaps)
- Software must ensure that the resultant DepthBuffer Coordinate X,Y values are non-negative.
- There are alignment restrictions – see 3DSTATE_DEPTH_BUFFER command.

8.1.4 Stencil Test

The Stencil Test function can be used to discard pixels based on a comparison between the [**Backface**] **Stencil Test Reference** state variable and the pixel's stencil value. This is a general purpose function used for such effects as shadow volumes, per-pixel clipping, etc. The result of this comparison is used in the Stencil Buffer Update function later in the pipeline.

This function is enabled by the **Stencil Test Enable** state variable. If ENABLED, the current stencil buffer value for this pixel is read.

Programming Notes:

- If the Depth Buffer is either undefined or does **not** have a surface format of D32_FLOAT_S8X24_UINT or D24_UNORM_S8_UINT and separate stencil buffer is disabled, **Stencil Test Enable** must be DISABLED.

A 2nd set of the stencil test state variables is provided so that pixels from back-facing objects, assuming they are not culled, can have a stencil test performed on them separate from the test for normal front-facing objects. The separate stencil test for back-facing objects can be enabled via the **Double Sided Stencil Enable** state variable. Otherwise, non-culled back-facing objects will use the same test function, mask and reference value as front-facing objects. The 2nd stencil state for back-facing objects is most commonly used to improve the performance of rendering shadow volumes which require a different stencil buffer operation depending on whether pixels rendered are from a front-facing or back-facing object. The backface stencil state removes the requirement to render the shadow volumes in 2 passes or sort the objects into front-facing and back-facing lists.

The remainder of this subsection describes the function in term of [**Backface**] **<state variable name>**. The Backface set of state variables are only used if Double Sided Stencil Enable is ENABLED and the object is considered back-facing. Otherwise the normal (front-facing) state variables are used.

This function then compares the [**Backface**] **Stencil Test Reference** value and the pixel's stencil value value after logically ANDing both values by [**Backface**] **Stencil Test Mask**. The comparison performed is specified by the [**Backface**] **Stencil Test Function** state variable. The result of the comparison is passed down the pipeline for use in the Stencil Buffer Update function. The Stencil Test function does not in itself discard pixels.

If **Stencil Test Enable** is DISABLED, a result of "stencil test passed" is propagated down the pipeline.

8.1.5 Depth Test

The Depth Test function can be used to discard pixels based on a comparison between the incoming pixel's depth value and the current depth buffer value associated with the pixel. This function is typically used to perform the "Z Buffer" hidden surface removal. The result of this pipeline function is used in the Stencil Buffer Update function later in the pipeline.

This function is enabled by the **Depth Test Enable** state variable. If enabled, the pixel's ("source") depth value is first computed. After computation the pixel's depth value is clamped to the range defined by **Minimum Depth** and **Maximum Depth** in the selected CC_VIEWPORT state. Then the current ("destination") depth buffer value for this pixel is read.

This function then compares the source and destination depth values. The comparison performed is specified by the **Depth Test Function** state variable.

The result of the comparison is propogated down the pipeline for use in the subsequent Depth Buffer Update function. The Depth Test function does not in itself discard pixels.

If **Depth Test Enable** is DISABLED, a result of "depth test passed" is propagated down the pipeline.

Programming Notes:

- Enabling the Depth Test function without defining a Depth Buffer is UNDEFINED.

8.1.6 Pre-Blend Color Clamping

Pre-Blend Color Clamping, controlled via **Pre-Blend Color Clamp Enable** and **Color Clamp Range** states in COLOR_CALC_STATE, is affected by the enabling of Color Buffer Blend as described below.

The following table summarizes the requirements involved with Pre-/Post-Blend Color Clamping.

Blending	RT Format	Pre-Blend Color Clamp	Post-Blend Color Clamp
Off	UNORM, UNORM_SRGB,YCRCB	Must be enabled with range = RT range or [0,1] (same function)	n/a, state ignored
	SNORM	Must be enabled with range = RT range or [-1,1] (same function)	n/a, state ignored
	FLOAT (except for R11G11B10_FLOAT)	Must be enabled (with any desired range)	n/a, state ignored
	R11G11B10_FLOAT	Must be enabled with either [0,1] or RT range	n/a, state ignored
	UINT, SINT	State ignored, implied clamp to RT range	n/a, state ignored
On (where permitted)	UNORM, UNORM_SRGB	Must be enabled with range = RT range or [0,1] (same function)	Must be enabled with range = RT range or [0,1] (same function)
	SNORM	Must be enabled with range = RT range or [-1,1] (same function)	Must be enabled with range = RT range or [-1,1] (same function)
	FLOAT (except for R11G11B10_FLOAT)	Can be disabled or enabled (with any desired range)	Must be enabled (with any desired range)
	R11G11B10_FLOAT	Can be disabled or enabled (with any desired range)	Must be enabled with either [0,1] or RT range

8.1.6.1.1 Pre-Blend Color Clamping when Blending is Disabled

The clamping of source color components is controlled by **Pre-Blend Color Clamp Enable**. If ENABLED, all source color components are clamped to the range specified by **Color Clamp Range**. If DISABLED, no clamping is performed.

Programming Notes:

- Given the possibility of writing UNPREDICTABLE values to the Color Buffer, it is expected and highly recommended that, when blending is disabled, software set **Pre-Blend Color Clamp Enable** to ENABLED and select an appropriate **Color Clamp Range**.
- When using SINT or UINT rendertarget surface formats, **Blending must** be DISABLED. The **Pre-Blend Color Clamp Enable** and **Color Clamp Range** fields are ignored, and an implied clamp to the rendertarget surface format is performed.

8.1.6.1.2 Pre-Blend Color Clamping when Blending is Enabled

The clamping of source, destination and constant color components is controlled by **Pre-Blend Color Clamp Enable**. If ENABLED, all these color components are clamped to the range specified by **Color Clamp Range**. If DISABLED, no clamping is performed on these color components prior to blending.

8.1.7 Color Buffer Blending

The Color Buffer Blending function is used to combine one or two incoming "source" pixel color+alpha values with the "destination" color+alpha read from the corresponding location in a RenderTarget.

Blending is enabled on a global basis by the **Color Buffer Blend Enable** state variable (in COLOR_CALC_STATE). If DISABLED, Blending and Post-Blend Clamp functions are disabled for <u>all</u> RenderTargets, and the pixel values (possibly subject to Pre-Blend Clamp) are passed through unchanged.

[DevSNB+]: The Color Buffer Blend Enable is in the per-render-target BLEND_STATE, and the field in SURFACE_STATE is no longer supported.

Programming Note:

- Color Buffer Blending and Logic Ops must not be enabled simultaneously, or behavior is UNDEFINED.
- Dual source blending:
  - **[DevCTG+]:** The DataPort only supports dual source blending with a SIMD8-style message.
- Only certain surface formats support Color Buffer Blending. Refer to the Surface Format tables in *Sampling Engine*. Blending must be disabled on a RenderTarget if blending is not supported.

The incoming "source" pixel values are modulated by a selected "source" blend factor, and the possibly gamma-decorrected "destination" values are modulated by a "destination" blend factor. These terms are then combined with a "blend function". In general:

src_term = src_blend_factor * src_color

dst_term = dst_blend_factor * dst_color

color output = blend_function(src_term, dst_term)

If there is no alpha value contained in the Color Buffer, a default value of 1.0 is used and, correspondingly, there is no alpha component computed by this function.

The blending of the color and alpha components is controlled with two separate (color and alpha) sets of state variables. However, if the **Independent Alpha Blend Enable** state variable in COLOR_CALC_STATE is DISABLED, then the "color" (rather than "alpha") set of state variables is used for both color and alpha. Note that this is the only use of the **Independent Alpha Blend Enable** state – it does not control whether Blending occurs, only how.

[DevSNB+] Per Render Target Blend State: Blend state is selected based on **Render Target Index** contained in the message header, and appropriate blend state is applied to Render Target Write messages.

The following table describes the color source and destination blend factors controlled by the **Source [Alpha] Blend Factor** and **Destination [Alpha] Blend Factor** state variables in COLOR_CALC_STATE. Note that the blend factors applied to the R,G,B channels are always controlled by the **Source/Destination Blend Factor**, while the blend factor applied to the alpha channel is controlled either by **Source/Destination Blend Factor** or **Source/Destination Alpha Blend Factor**.

Table 23. Color Buffer Blend Color Factors

Blend Factor Selection	Blend Factor Applied for R,G,B,A channels (oN = output from PS to RT#N) (o1 = 2nd output from PS in Dual-Souce mode only) (rtN = destination color from RT#N) (CC = Constant Color)
BLENDFACTOR_ZERO	0.0, 0.0, 0.0, 0.0
BLENDFACTOR_ONE	1.0, 1.0, 1.0, 1.0
BLENDFACTOR_SRC_COLOR	oN.r, oN.g, oN.b, oN.a
BLENDFACTOR_INV_SRC_COLOR	1.0-oN.r, 1.0-oN.g, 1.0-oN.b, 1.0-oN.a
BLENDFACTOR_SRC_ALPHA	oN.a, oN.a, oN.a, oN.a
BLENDFACTOR_INV_SRC_ALPHA	1.0-oN.a, 1.0-oN.a, 1.0-oN.a, 1.0-oN.a
BLENDFACTOR_SRC1_COLOR	o1.r, o1.g, o1.b, o1.a
BLENDFACTOR_INV_SRC1_COLOR	1.0-o1.r, 1.0-o1.g, 1.0-o1.b, 1.0-o1.a
BLENDFACTOR_SRC1_ALPHA	o1.a, o1.a, o1.a, o1.a
BLENDFACTOR_INV_SRC1_ALPHA	1.0-o1.a, 1.0-o1.a, 1.0-o1.a, 1.0-o1.a
BLENDFACTOR_DST_COLOR	rtN.r, rtN.g, rtN.b, rtN.a
BLENDFACTOR_INV_DST_COLOR	1.0-rtN.r, 1.0-rtN.g, 1.0-rtN.b, 1.0-rtN.a
BLENDFACTOR_DST_ALPHA	rtN.a, rtN.a, rtN.a, rtN.a
BLENDFACTOR_INV_DST_ALPHA	1.0-rtN.a, 1.0-rtN.a, 1.0-rtN.a, 1.0-rtN.a

Blend Factor Selection	Blend Factor Applied for R,G,B,A channels (oN = output from PS to RT#N) (o1 = 2nd output from PS in Dual-Souce mode only) (rtN = destination color from RT#N) (CC = Constant Color)
BLENDFACTOR_CONST_COLOR	CC.r, CC.g, CC.b, CC.a
BLENDFACTOR_INV_CONST_COLOR	1.0-CC.r, 1.0-CC.g, 1.0-CC.b, 1.0-CC.a
BLENDFACTOR_CONST_ALPHA	CC.a, CC.a, CC.a, CC.a
BLENDFACTOR_INV_CONST_ALPHA	1.0-CC.a, 1.0-CC.a, 1.0-CC.a, 1.0-CC.a
BLENDFACTOR_SRC_ALPHA_SATURATE	f,f,f,1.0 where f = min(1.0 – rtN.a, oN.a)

The following table lists the supported blending operations defined by the **Color Blend Function** state variable and the **Alpha Blend Function** state variable (when in independent alpha blend mode).

Table 24. Color Buffer Blend Functions

Blend Function	Operation (for each color component)
BLENDFUNCTION_ADD	SrcColor*SrcFactor + DstColor*DstFactor
BLENDFUNCTION_SUBTRACT	SrcColor*SrcFactor - DstColor*DstFactor
BLENDFUNCTION_REVERSE_ SUBTRACT	DstColor*DstFactor - SrcColor*SrcFactor
BLENDFUNCTION_MIN	min (SrcColor*SrcFactor, DstColor*DstFactor) **Programming Note:** This is a superset of the OpenGL "min" function.
BLENDFUNCTION_MAX	max (SrcColor*SrcFactor, DstColor*DstFactor) **Programming Note:** This is a superset of the OpenGL "max" function.

8.1.8 Post-Blend Color Clamping

(See *Pre-Blend Color Clamping* above for a summary table regarding clamping)

Post-Blend Color clamping is available only if Blending is enabled.

If Blending is enabled, the clamping of blending output color components is controlled by **Post-Blend Color Clamp Enable**. If ENABLED, the color components output from blending are clamped to the range specified by **Color Clamp Range**. If DISABLED, no clamping is performed at this point.

Regardless of the setting of **Post-Blend Color Clamp Enable**, when Blending is enabled color components will be automatically clamped to (at least) the rendertarget surface format range at this stage of the pipeline.

8.1.9 Color Quantization

[This is considered an implementation-specific topic, covered in the detailed hardware design documents]

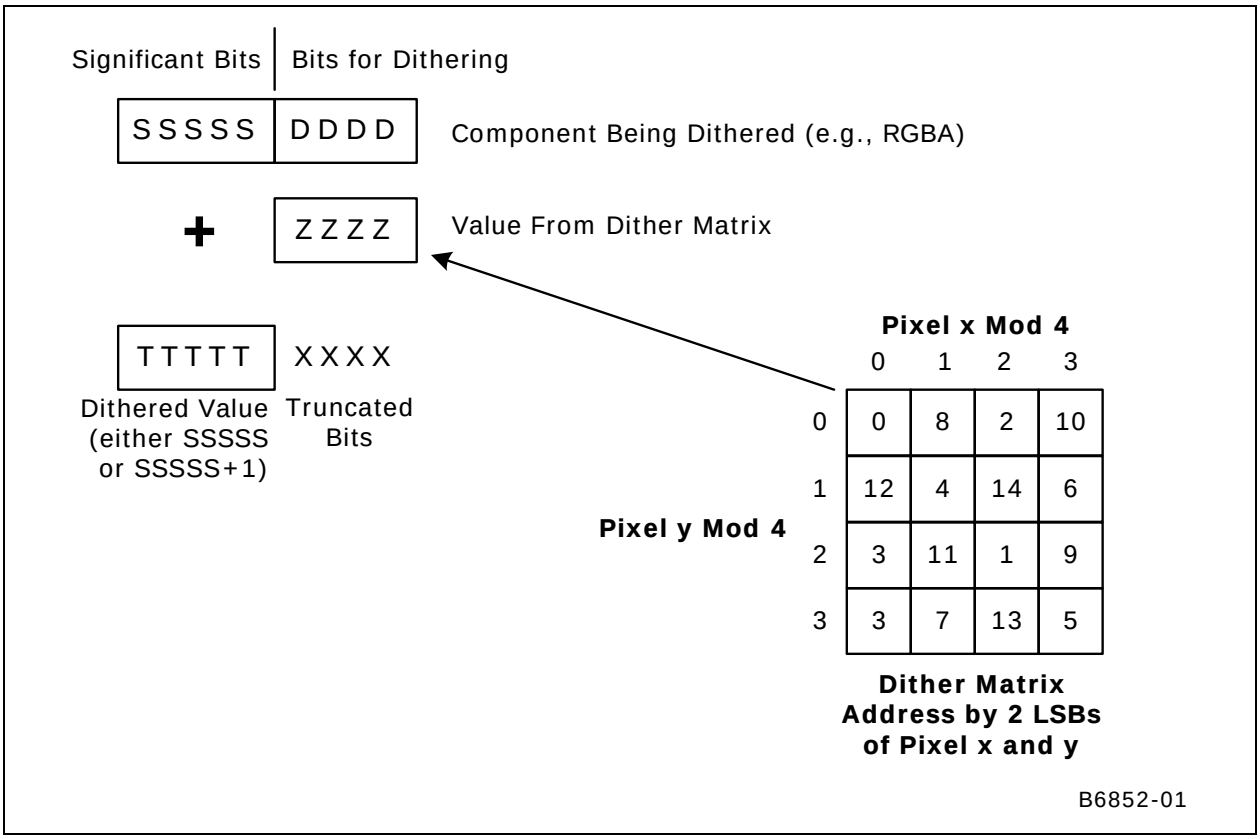
8.1.10 Dithering

Dithering is used to give the illusion of a higher resolution when using low-bpp channels in color buffers (e.g., with 16bpp color buffer). By carefully choosing an arrangement of lower resolution colors, colors otherwise not representable can be approximated, especially when seen at a distance where the viewer's eyes will average adjacent pixel colors. Color dithering tends to diffuse the sharp color bands seen on smooth-shaded objects.

A four-bit dither value is obtained from a 4x4 Dither Constant matrix depending on the pixel's X and Y screen coordinate. The pixel's X and Y screen coordinates are first offset by the **Dither Offset X** and **Dither Offset Y** state variables (these offsets are used to provide window-relative dithering). Then the two LSBs of the pixel's screen X coordinate are used to address a column in the dither matrix, and the two LSBs of the pixel's screen Y coordinate are used to address a row. This way, the matrix repeats every four pixels in both directions.

The value obtained is appropriately shifted to align with (what would be otherwise) truncated bits of the component being dithered. It is then added with the component and the result is truncated to the bit depth of the component given the color buffer format.

Figure 8-1. Dithering Process (5-Bit Example)



8.1.11 Logic Ops

The Logic Ops function is used to combine the incoming “source” pixel color/alpha values with the corresponding “destination” color/alpha contained in the ColorBuffer, using a logic function.

The Logic Op function is enabled by the **LogicOp Enable** state variable. If DISABLED, this function is ignored and the incoming pixel values are passed through unchanged.

Programming Note:

- Color Buffer Blending and Logic Ops must not be enabled simultaneously, or behavior is UNDEFINED.
- Logic Ops are only supported on *_UNORM surfaces (excluding _SRGB variants), otherwise Logic Ops must be DISABLED.

The following table lists the supported logic ops. The logic op is selected using the **Logic Op Function** field in COLOR_CALC_STATE.

Table 25. Logic Ops

LogicOp Function	Definition (S=Source, D=Destination)
LOGICOP_CLEAR	all 0's
LOGICOP_NOR	NOT (S OR D)
LOGICOP_AND_INVERTED	(NOT S) AND D
LOGICOP_COPY_INVERTED	NOT S
LOGICOP_AND_REVERSE	S AND NOT D
LOGICOP_INVERT	NOT D
LOGICOP_XOR	S XOR D
LOGICOP_NAND	NOT (S AND D)
LOGICOP_AND	S AND D
LOGICOP_EQUIV	NOT (S XOR D)
LOGICOP_NOOP	D
LOGICOP_OR_INVERTED	(NOT S) OR D
LOGICOP_COPY	S
LOGICOP_OR_REVERSE	S OR NOT D
LOGICOP_OR	S OR D
LOGICOP_SET	all 1's

8.1.12 Buffer Update

The Buffer Update function is responsible for updating the pixel's Stencil, Depth and Color Buffer contents based upon the results of the Stencil and Depth Test functions. Note that Kill Pixel and/or Alpha Test functions may have already discarded the pixel by this point.

8.1.12.1 Stencil Buffer Updates

If and only if stencil testing is enabled, the Stencil Buffer is updated according to the **Stencil Fail Op**, **Stencil Pass Depth Fail Op**, and **Stencil Pass Depth Pass Op** state (or their backface counterparts if **Double Sided Stencil Enable** is ENABLED and the pixel is from a back-facing object) and the results of the Stencil Test and Depth Test functions.

Stencil Fail Op and **Backface Stencil Fail Op** specify how/if the stencil buffer is modified <u>if the stencil test fails</u>. **Stencil Pass Depth Fail Op** and **Backface Stencil Pass Depth Fail Op** specify how/if the stencil buffer is modified <u>if the stencil test passes but the depth test fails</u>. **Stencil Pass Depth Pass Op** and **Backface Stencil Pass Depth Pass Op** specify how/if the stencil buffer is modified <u>if both the stencil and depth tests pass</u>. The operations (on the stencil buffer) that are to be performed under one of these (mutually exclusive) conditions is summarized in the following table.

Table 26. Stencil Buffer Operations

Stencil Operation	Description
STENCILOP_KEEP	Do not modify the stencil buffer
STENCILOP_ZERO	Store a 0
STENCILOP_REPLACE	Store the *StencilTestReference* reference value
STENCILOP_INCRSAT	Saturating increment (clamp to max value)
STENCILOP_DECRSAT	Saturating decrement (clamp to 0)
STENCILOP_INCR	Increment (possible wrap around to 0)
STENCILOP_DECR	Decrement (possible wrap to max value)
STENCILOP_INVERT	Logically invert the stencil value

Any and all writes to the stencil portion of the depth buffer are enabled by the **Stencil Buffer Write Enable** state variable.

When writes are enabled, the **Stencil Buffer Write Mask** and **Backface Stencil Buffer Write Mask** state variables provide an 8-bit mask that selects which bits of the stencil write value are modified. Masked-off bits (i.e., mask bit == 0) are left unmodified in the Stencil Buffer.

Programming Notes:

- If the Depth Buffer does **not** have a surface format of D32_FLOAT_S8X24_UINT or D24_UNORM_S8_UINT, **Stencil Buffer Write Enable** must be DISABLED.
- The Stencil Buffer can be written even if depth buffer writes are disabled via **Depth Buffer Write Enable**.

8.1.12.2 Depth Buffer Updates

Any and all writes to the Depth Buffer are enabled by the **Depth Buffer Write Enable** state variable. If there is no Depth Buffer, writes must be explicitly disabled with this state variable, or operation is UNDEFINED.

If depth testing is disabled or the depth test passed, the incoming pixel's depth value is written to the Depth Buffer. If depth testing is enabled and the depth test failed, the pixel is discarded – with no modification to the Depth or Color Buffers (though the Stencil Buffer may have been modified).

8.1.12.3 Color Gamma Correction

Computed RGB (not A) channels can be gamma-corrected prior to update of the Color Buffer.

This function is automatically invoked whenever the destination surface (render target) has an SRGB format (see surface formats in *Sampling Engine*). For these surfaces, the computed RGB values are converted from gamma=1.0 space to gamma=2.4 space by applying a ^(2.4) exponential function.

8.1.12.4 Color Buffer Updates

Finally, if the pixel has not been discarded by this point, the incoming pixel color is written into the Color Buffer. The **Surface Format** of the color buffer indicates which channel(s) are written (e.g., R8G8_UNORM are written with the Red and Green channels only). The **Color Buffer Component Write Disables** from the Color Buffer's SURFACE_STATE provide an independent write disable for each channel of the Color Buffer.

8.2 Pixel Pipeline State Summary

8.2.1 COLOR_CALC_STATE

8.2.1.1 COLOR_CALC_STATE [DevSNB+]

COLOR_CALC_STATE		
Project: DevSNB+		
This definition applies to [DevSNB+] devices. It is pointed to by a field in 3DSTATE_CC_STATE_POINTERS, and stored at a 64-byte aligned boundary.		
DWord	**Bit**	**Description**
0	31:24	Stencil Reference Value Format: U8.0 This field specifies the stencil reference value to compare against in the (front face) StencilTest function.
	23:16	BackFace Stencil Reference Value Format: U8.0 This field specifies the stencil reference value to compare against in the StencilTest function.
	15	Round Disable Function Disable Project: All Format: U8.0 Disables the round-disable function of the color calculator. If this bit is zero, dithering is cancelled based on the data used by blend to avoid drift. If this bit is one, this is not done.
	14:1	Reserved Project: All Format: MBZ

COLOR_CALC_STATE

	0	Alpha Test Format Project: All This field selects the format for Alpha Reference Value and the format in which Alpha Test is performed.

Value	Name	Description	Project
0h	ALPHATEST_UNORM8	UNorm8	All
1h	ALPHATEST_FLOAT32	Float32	All

Programming Notes
Alpha-test format is independent of RT format. When PS outputs UNIT/SINT alpha-value, it will be treated as IEEE 32bit float number for the purpose of alpha-test.

1	31:0	Alpha Reference Value Project: All Exists If: **Alpha Test Format** == ALPHATEST_UNORM8 Format: UNORM8 Upper 24 bits MBZ This field specifies the alpha reference value to compare against in the Alpha Test function.
	31:0	Alpha Reference Value Project: All Exists If: **Alpha Test Format** == ALPHATEST_ FLOAT32 Format: IEEE_Float This field specifies the alpha reference value to compare against in the Alpha Test function.
2	31:0	Blend Constant Color Red Format: IEEE_Float This field specifies the Red channel of the Constant Color used in Color Buffer Blending.
3	31:0	Blend Constant Color Green Format: IEEE_Float This field specifies the Green channel of the Constant Color used in Color Buffer Blending.
4	31:0	Blend Constant Color Blue Format: IEEE_Float This field specifies the Blue channel of the Constant Color used in Color Buffer Blending.

COLOR_CALC_STATE		
5	31:0	Blend Constant Color Alpha Format: IEEE_Float This field specifies the Alpha channel of the Constant Color used in Color Buffer Blending.

8.2.2 DEPTH_STENCIL_STATE [DevSNB+]

DEPTH_STENCIL_STATE		
Project: DevSNB+		
The DEPTH_STENCIL_STATE is pointed to by a field in 3DSTATE_CC_STATE_POINTERS. It is stored at a 64-byte aligned boundary.		
DWord	**Bit**	**Description**
0	31	Stencil Test Enable Project: All Format: Enable Enables StencilTest function of the Pixel Processing pipeline. **Programming Notes** If any of the render targets are YUV format, this field must be disabled. This field cannot be enabled if **Surface Format** in 3DSTATE_DEPTH_BUFFER is set to D16_UNORM.
	30:28	Stencil Test Function Project: All Format: 3D_CompareFunction This field specifies the comparison function used in the (front face) StencilTest function.

Value	Name	Description	Project
0h	COMPAREFUNCTION_ALWAYS		All
1h	COMPAREFUNCTION_NEVER		All
2h	COMPAREFUNCTION_LESS		All
3h	COMPAREFUNCTION_EQUAL		All
4h	COMPAREFUNCTION_LEQUAL		All
5h	COMPAREFUNCTION_GREATER		All
6h	COMPAREFUNCTION_NOTEQUAL		All
7h	COMPAREFUNCTION_GEQUAL		All

DEPTH_STENCIL_STATE

27:25 Stencil Fail Op

Project: All

Format: 3D_StencilOperation

This field specifies the operation to perform on the Stencil Buffer when the (front face) stencil test fails.

Note: if all three stencil ops (Stencil Fail, Stencil Pass Depth Fail, and Stencil Pass Depth Pass) are KEEP, ZERO, or REPLACE, the stencil buffer is not read.

Value	Name	Description	Project
0	STENCILOP_KEEP		All
1	STENCILOP_ZERO		All
2	STENCILOP_REPLACE		All
3	STENCILOP_INCRSAT		All
4	STENCILOP_DECRSAT		All
5	STENCILOP_INCR		All
6	STENCILOP_DECR		All
7	STENCILOP_INVERT		All

24:22 Stencil Pass Depth Fail Op

Project: All

Format: 3D_StencilOperation see Stencil Fail Op

This field specifies the operation to perform on the Stencil Buffer when the (front face) stencil test passes but the depth pass fails.

21:19 Stencil Pass Depth Pass Op

Project: All

Format: 3D_StencilOperation see Stencil Fail Op

This field specifies the operation to perform on the Stencil Buffer when the (front face) stencil test passes and the depth pass passes (or is disabled).

18 Stencil Buffer Write Enable

Project: All

Format: Enable

Enables writes to the Stencil Buffer. If **Stencil Test Enable** is disabled, writes to the stencil buffer are disabled independent of the setting of this field.

Programming Notes
This field cannot be enabled if **Surface Format** in 3DSTATE_DEPTH_BUFFER is set to D16_UNORM.

DEPTH_STENCIL_STATE

	17:16	Reserved Project: All Format: MBZ
	15	Double Sided Stencil Enable Project: All Format: Enable Enable doubled sided stencil operations.

Value	Name	Description	Project
1	TRUE	Double Sided Stencil Enabled	All
0	FALSE	Double Sided Stencil Disabled	All

Programming Notes
Back-facing primitives have a vertex winding order opposite to the currently selected **Front Winding** state. Culling of primitives is not affected by the double sided stencil state Back-facing primitives will be rendered, honoring all current device state, as though it were a front-facing primitive with no implicitly overloaded state.

	14:12	**BackFace Stencil Test Function** Project: All Format: 3D_CompareFunction This field specifies the comparison function used in the StencilTest function.

Value	Name	Description	Project
0h	COMPAREFUNCTION_ALWAYS		All
1h	COMPAREFUNCTION_NEVER		All
2h	COMPAREFUNCTION_LESS		All
3h	COMPAREFUNCTION_EQUAL		All
4h	COMPAREFUNCTION_LEQUAL		All
5h	COMPAREFUNCTION_GREATER		All
6h	COMPAREFUNCTION_NOTEQUAL		All
7h	COMPAREFUNCTION_GEQUAL		All

DEPTH_STENCIL_STATE

	11:9	**Backface Stencil Fail Op** Project: All Format: 3D_StencilOperation This field specifies the operation to perform on the Stencil Buffer when the stencil test fails.

Value	Name	Description	Project
0	STENCILOP_KEEP	STENCILOP_KEEP	All
1	STENCILOP_ZERO	STENCILOP_ZERO	All
2	STENCILOP_REPLACE	STENCILOP_REPLACE	All
3	STENCILOP_INCRSAT	STENCILOP_INCRSAT	All
4	STENCILOP_DECRSAT	STENCILOP_DECRSAT	All
5	STENCILOP_INCR	STENCILOP_INCR	All
6	STENCILOP_DECR	STENCILOP_DECR	All
7	STENCILOP_INVERT	STENCILOP_INVERT	All

	8:6	**Backface Stencil Pass Depth Fail Op** Project: All Format: 3D_StencilOperation see Stencil Fail Op This field specifies the operation to perform on the Stencil Buffer when the stencil test passes but the depth pass fails.
	5:3	**Backface Stencil Pass Depth Pass Op** Project: All Format: 3D_StencilOperation see Stencil Fail Op This field specifies the operation to perform on the Stencil Buffer when the stencil test passes and the depth pass passes (or is disabled).
	2:0	Reserved Project: All Format: MBZ
1	31:24	**Stencil Test Mask** Project: All Format: U8 This field specifies a bit mask applied to stencil test values. Both the stencil reference value and value read from the stencil buffer will be logically ANDed with this mask before the stencil comparison test is performed.
	23:16	**Stencil Write Mask** Project: All Format: U8 This field specifies a bit mask applied to stencil buffer writes. Only those stencil buffer bits corresponding to bits set in this mask will be modified.

DEPTH_STENCIL_STATE

	15:8	**Backface Stencil Test Mask** Project: All Format: U8 This field specifies a bit mask applied to backface stencil test values. Both the stencil reference value and value read from the stencil buffer will be logically ANDed with this mask before the stencil comparison test is performed.
	7:0	**Backface Stencil Write Mask** Project: All Format: U8 This field specifies a bit mask applied to backface stencil buffer writes. Only those stencil buffer bits corresponding to bits set in this mask will be modified.
2	31	**Depth Test Enable** Project: All Format: Enable Enables the DepthTest function of the Pixel Processing pipeline.
	30	Reserved Project: All Format: MBZ

Programming Notes
If any of the render targets are YUV format, this field must be disabled.

Errata	Description	Project
	Software must issue a PIPE_CONTROL command with the **Write Cache Flush Enable** set before transitioning from write-only depth/stencil mode (**Depth Test Enable** and **Stencil Test Enable** both DISABLED and **Depth Buffer Write Enable** or **Stencil Buffer Write Enable** ENABLED) to read/write depth/stencil mode (**Depth Test Enable** or **Stencil Test Enable** ENABLED), otherwise operation is UNDEFINED.	DevBW, DevCL-A

DEPTH_STENCIL_STATE

29:27

Depth Test Function

Project: All

Format: 3D_DepthTestFunction

Specifies the comparison function used in DepthTest function.

Value	Name	Description	Project
0h	COMPAREFUNCTION_ALWAYS		All
1h	COMPAREFUNCTION_NEVER		All
2h	COMPAREFUNCTION_LESS		All
3h	COMPAREFUNCTION_EQUAL		All
4h	COMPAREFUNCTION_LEQUAL		All
5h	COMPAREFUNCTION_GREATER		All
6h	COMPAREFUNCTION_NOTEQUAL		All
7h	COMPAREFUNCTION_GEQUAL		All

Programming Notes
if the Depth Test Function is ALWAYS or NEVER, the depth buffer is not read.

26

Depth Buffer Write Enable

Project: All

Format: Enable

Enables writes to the Depth Buffer.

Programming Notes
A Depth Buffer must be defined before enabling writes to it, or operation is UNDEFINED.

Errata	Description	Project
	See relevant errata in **Depth Test Enable** above	DevBW,DevCL-A

25:0 Reserved Project: All Format: MBZ

8.2.3 BLEND_STATE [DevSNB+]

BLEND_STATE

BLEND_STATE

Project: DevSNB+

The blend state is stored as an array of up to 8 elements, each of which contains the DWords described here. The start of each element is spaced 2 DWords apart. The first element of the blend state array is aligned to a 64-byte boundary, which is pointed to by a field in 3DSTATE_CC_STATE_POINTERS. The 3-bit **Render Target Index** field in the Render Target Write data port message header is used to select which of the 8 elements from BLEND_STATE that is used on the current message.

DWord	Bit	Description
0	31	**Color Buffer Blend Enable** Project: All Format: Enable Enables the ColorBufferBlending (nee "alpha blending") function of the Pixel Processing Pipeline for this render target. **Programming Notes** Enabling LogicOp and ColorBufferBlending at the same time is UNDEFINED
	30	**Independent Alpha Blend Enable** Project: All Format: Enable When enabled, the other fields in this instruction control the combination of the alpha components in the Color Buffer Blend stage. When disabled, the alpha components are combined in the same fashion as the color components.
	29	Reserved Project: All Format: MBZ
	28:26	**Alpha Blend Function** Project: All Format: 3D_ColorBufferBlendFunction This field specifies the function used to combine the alpha components in the Color Buffer blend stage of the Pixel Pipeline when the *IndependentAlphaBlend* state is enabled.

Value	Name	Description	Project
0	BLENDFUNCTION_ADD		All
1	BLENDFUNCTION_SUBTRACT		All
2	BLENDFUNCTION_REVERSE_SUBTRACT		All
3	BLENDFUNCTION_MIN		All
4	BLENDFUNCTION_MAX		All
5 - 7	Reserved		All

	25	Reserved Project: All Format: MBZ

BLEND_STATE

24:20

Source Alpha Blend Factor

Project: All

Format: 3D_ColorBufferBlendFactor

Controls the “source factor” in alpha Color Buffer Blending stage.

Note: For the source/destination alpha blend factors, the encodings indicating “COLOR” are the same as the encodings indicating “ALPHA”, as the alpha component of the color is selected.

Value	Name	Description	Project
00h	Reserved		All
01h	BLENDFACTOR_ONE		All
02h	BLENDFACTOR_SRC_COLOR		All
03h	BLENDFACTOR_SRC_ALPHA		All
04h	BLENDFACTOR_DST_ALPHA		All
05h	BLENDFACTOR_DST_COLOR		All
06h	BLENDFACTOR_SRC_ALPHA_SATURATE		All
07h	BLENDFACTOR_CONST_COLOR		All
08h	BLENDFACTOR_CONST_ALPHA		All
09h	BLENDFACTOR_SRC1_COLOR		All
0Ah	BLENDFACTOR_SRC1_ALPHA		All
0Bh-10h	Reserved		All
11h	BLENDFACTOR_ZERO		All
12h	BLENDFACTOR_INV_SRC_COLOR		All
13h	BLENDFACTOR_INV_SRC_ALPHA		All
14h	BLENDFACTOR_INV_DST_ALPHA		All
15h	BLENDFACTOR_INV_DST_COLOR		All
16h	Reserved		All
17h	BLENDFACTOR_INV_CONST_COLOR		All
18h	BLENDFACTOR_INV_CONST_ALPHA		All
19h	BLENDFACTOR_INV_SRC1_COLOR		All
1Ah	BLENDFACTOR_INV_SRC1_ALPHA		All

19:15

Destination Alpha Blend Factor

Project: All

Format: 3D_ColorBufferBlendFactor

Controls the “destination factor” in alpha Color Buffer Blending stage.

Refer to **Source Alpha Blend Factor** for encodings.

BLEND_STATE

	14	Reserved Project: All Format: MBZ
	13:11	**Color Blend Function** Project: All Format: 3D_ColorBufferBlendFunction This field specifies the function used to combine the color components in the ColorBufferBlending function of the Pixel Processing Pipeline. If **Independent Alpha Blend Enable** is disabled, this field will also control the blending of the alpha components in the ColorBufferBlending function.

Value	Name	Description	Project
0	BLENDFUNCTION_ADD		All
1	BLENDFUNCTION_SUBTRACT		All
2	BLENDFUNCTION_REVERSE_SUBTRACT		All
3	BLENDFUNCTION_MIN		All
4	BLENDFUNCTION_MAX		All

	10	Reserved Project: All Format: MBZ
	9:5	Reserved
1	31	**AlphaToCoverage Enable** Project: All Format: Enable If set, Source0 Alpha is converted to a temporary 1/2/4-bit coverage mask and the mask bit corresponding to the sample# ANDed with the sample mask bit. If set, sample coverage is computed based on src0 alpha value. Value of 0 disables all samples and value of 1 enables all samples for that pixel. The same coverage needs to apply to all the RTs in MRT case. Further, any value of src0 alpha between 0 and 1 monotonically increases the number of enabled pixels. The same coverage needs to be applied to all the RTs in MRT case. **[DevSNB-A] Errata:** This bit must be disabled.
	30	**AlphaToOne Enable** Project: All Format: Enable If set, Source0 Alpha is set to 1.0f after (possibly) being used to generate the AlphaToCoverage coverage mask. The same coverage needs to be applied to all the RTs in MRT case. If Dual Source Blending is enabled, this bit must be disabled. **[DevSNB-A] Errata:** This bit must be disabled.

BLEND_STATE

29 — AlphaToCoverage Dither Enable

Project: All

Format: Enable

If set, sample coverage is computed based on src0 alpha value and it modulates the sample coverage based on screen coordinates. Value of 0 disables all samples and value of 1 enables all samples for that pixel. The same coverage needs to apply to all the RTs in MRT case. Further, any value of src0 alpha between 0 and 1 monotonically increases the number of enabled pixels.

The same coverage needs to be applied to all the RTs in MRT case.

If AlphaToCoverage is disabled, AlphaToCoverage Dither does not have any impact.

[DevSNB]: This bit must be disabled.

This bit is not supported on **[DevSNB:A]** and **[DevSNB:B]**

28 — Reserved Project: All Format: MBZ

27 — Write Disable Alpha

Project: All

Format: Disable

This field controls the writing of the alpha component into the Render Target.

Value	Name	Description	Project
0B	Enabled	Alpha component can be overwritten	All
1B	Disabled	Writes to the color buffer will not modify Alpha.	All

Programming Notes
For YUV surfaces, this field must be set to 0B (enabled).

26 — Write Disable Red

Project: All

Format: Disable

This field controls the writing of the red component into the Render Target.

Value	Name	Description	Project
0B	Enabled	Red component can be overwritten	All
1B	Disabled	Writes to the color buffer will not modify Red.	All

Programming Notes
For YUV surfaces, this field must be set to 0B (enabled).

BLEND_STATE

25 Write Disable Green

Project: All

Format: Disable

This field controls the writing of the green component into the Render Target.

Value	Name	Description	Project
0B	Enabled	Green component can be overwritten	All
1B	Disabled	Writes to the color buffer will not modify Green.	All

Programming Notes
For YUV surfaces, this field must be set to 0B (enabled).

24 **Write Disable Blue**

Project: All

Format: Disable

This field controls the writing of the Blue component into the Render Target.

Value	**Name**	**Description**	**Project**
0B	Enabled	Blue component can be overwritten	All
1B	Disabled	Writes to the color buffer will not modify Blue.	All

Programming Notes
For YUV surfaces, this field must be set to 0B (enabled).

23 **Reserved** Project: All Format: MBZ

22 **Logic Op Enable**

Project: All

Format: Enable

Enables the LogicOp function of the Pixel Processing pipeline.

Programming Notes
Enabling LogicOp and Color Buffer Blending at the same time is UNDEFINED

BLEND_STATE

	21:18	**Logic Op Function** Project: All Format: 3D_LogicOpFunction This field specifies the function to be performed (when enabled) in the Logic Op stage of the Pixel Processing pipeline. Note that the encoding of this field is one less than the corresponding "R2_" ROP code defined in WINGDI.H, and is a rather contorted mapping of the OpenGL LogicOp encodings. However, this field was defined such that, when the 4 bits are replicated to 8 bits, they coincide with the ROP codes used in the Blter. **Note:** if the Logic Op Function does not depend on "D", the dest buffer is not read.
	17	**Reserved** Project: All Format: MBZ

Value	Name	Description	Project
0h	LOGICOP_CLEAR	BLACK; all 0's	All
1h	LOGICOP_NOR	NOTMERGEPEN; NOT (S OR D)	All
2h	LOGICOP_AND_INVERTED	MASKNOTPEN; (NOT S) AND D	All
3h	LOGICOP_COPY_INVERTED	NOTCOPYPEN; NOT S	All
4h	LOGICOP_AND_REVERSE	MASKPENNOT; S AND NOT D	All
5h	LOGICOP_INVERT	NOT; NOT D	All
6h	LOGICOP_XOR	XORPEN; S XOR D	All
7h	LOGICOP_NAND	NOTMASKPEN; NOT (S AND D)	All
8h	LOGICOP_AND	MASKPEN; S AND D	All
9h	LOGICOP_EQUIV	NOTXORPEN; NOT (S XOR D)	All
Ah	LOGICOP_NOOP	NOP; D	All
Bh	LOGICOP_OR_INVERTED	MERGENOTPEN; (NOT S) OR D	All
Ch	LOGICOP_COPY	COPYPEN; S	All
Dh	LOGICOP_OR_REVERSE	MERGEPENNOT; S OR NOT D	All
Eh	LOGICOP_OR	MERGEPEN; S OR D	All
Fh	LOGICOP_SET	WHITE; all 1's	All

BLEND_STATE

16	**Alpha Test Enable** Project: All Format: Enable Enables the AlphaTest function of the Pixel Processing pipeline. **Programming Notes** Alpha Test can only be enabled if Pixel Shader outputs a float alpha value. Alpha Test is applied independently on each render target by comparing that render target's alpha value against the alpha reference value. If the alpha test fails, the corresponding pixel write will be supressed only for that render target. The depth/stencil update will occur if alpha test passes for *any* render target.
15:13	**Alpha Test Function** Project: All Format: 3D_CompareFunction This field specifies the comparison function used in the AlphaTest function

Value	Name	Description	Project
0h	COMPAREFUNCTION_ALWAYS	Always pass	All
1h	COMPAREFUNCTION_NEVER	Never pass	All
2h	COMPAREFUNCTION_LESS	Pass if the value is less than the reference	All
3h	COMPAREFUNCTION_EQUAL	Pass if the value is equal to the reference	All
4h	COMPAREFUNCTION_LEQUAL	Pass if the value is less than or equal to the reference	All
5h	COMPAREFUNCTION_GREATER	Pass if the value is greater than the reference	All
6h	COMPAREFUNCTION_NOTEQUAL	Pass if the value is not equal to the reference	All
7h	COMPAREFUNCTION_GEQUAL	Pass if the value is greater than or equal to the reference	All

12	**Color Dither Enable** Project: All Format: Enable Enables dithering of colors (including any alpha component) before they are written to the Color Buffer.

BLEND_STATE

	11:10	**X Dither Offset** Project: All Format: U2 Specifies offset to apply to pixel X coordinate LSBs when accessing dither table.
	9:8	**Y Dither Offset** Project: All Format: U2 Specifies offset to apply to pixel Y coordinate LSBs when accessing dither table.
	7:4	**Reserved** Project: All Format: MBZ
	3:2	**Color Clamp Range** Project: All Specifies the clamped range used in Pre-Blend and Post-Blend Color Clamp functions if one or both of those functions are enabled. Note that this range selection is shared between those functions. This field is ignored if both of the **Color Clamp Enables** are disabled

Value	Name	Description	Project
0	COLORCLAMP_UNORM	Clamp Range [0,1]	All
1	COLORCLAMP_SNORM	Clamp Range [-1,1]	All
2	COLORCLAMP_RTFORMAT	Clamp to the range of the RT surface format (Note: The Alpha component is clamped to FLOAT16 for R11G11B10_FLOAT format).	All
3	Reserved	Reserved	All

BLEND_STATE

	1	**Pre-Blend Color Clamp Enable** Project: All Format: Enable FormatDesc This field specifies whether the source, destination and constant color channels are clamped prior to blending, regardless of whether blending is enabled. If DISABLED, no clamping is performed prior to blending. If ENABLED, all inputs to the blend function are clamped prior to the blend to the range specified by **Color Clamp Range**.
	0	**Post-Blend Color Clamp Enable** Project: All Format: Enable If blending is enabled, this field specifies whether the blending output channels are first clamped to the range specified by **Color Clamp Range**. Regardless of whether this clamping is enabled, the blending output channels will be clamped to the RT surface format just prior to being written.

Value	Name	Description	Project
0	Disabled	No clamping is performed prior to blending.	All
1	Enabled	All inputs to the blend function are clamped prior to the blend to the range specified by **Color Clamp Range**.	All

Programming Notes

See table in *Pre-Blending Color Clamp* subsection for programming restrictions as a function of RT format.

This field is ignored (treated as DISABLED) for UINT and SINT RT surface formats. Blending is not supported for those RT surface formats. The device will automatically clamp source color channels to the respective RT surface range.

Programming Notes

See table in *Pre-Blending Color Clamp* subsection for programming restrictions as a function of RT format.

This field is ignored (treated as DISABLED) for UINT and SINT RT surface formats. Blending is not supported for those RT surface formats. The device will automatically clamp source color channels to the respective RT surface range.

Programming Note: CC Unit also receives 3DSTATE_WM_HZ_OP and 3DSTATE_PS_EXTRA.

8.2.4 CC_VIEWPORT

CC_VIEWPORT		
Project: All		
The viewport state is stored as an array of up to 16 elements, each of which contains the DWords described here. The start of each element is spaced 2 DWords apart. The first element of the viewport state array is aligned to a 32-byte boundary.		
DWord	**Bit**	**Description**
0	31:0	**Minimum Depth** Project: All Format: IEEE_Float FormatDesc Indicates the minimum depth. The interpolated or computed depth is clamped to this value prior to the depth test.
1	31:0	**Maximum Depth** Project: All Format: IEEE_Float FormatDesc Indicates the maximum depth. The interpolated or computed depth is clamped to this value prior to the depth test.

8.3 Other Pixel Pipeline Functions

8.3.1 Statistics Gathering

[DevSNB+]: If **Statistics Enable** is set in 3DSTATE_WM, the PS_DEPTH_COUNT register (see Memory Interface Registers in Volume Ia, *GPU*) will be incremented once for each pixel (or sample) that passes the depth, stencil and alpha tests. Note that each of these tests is treated as passing if disabled. This count is accurate regardless of whether **Early Depth Test Enable** is set. In order to obtain the value from this register at a deterministic place in the primitive stream without flushing the pipeline, however, the PIPE_CONTROL command must be used. See the *3D Pipeline* chapter in this volume for details on PIPE_CONTROL.

Revision History

Revision Number	Description	Revision Date
1.0	First 2011 OpenSource edition	May 2011

§§

Intel® OpenSource HD Graphics Programmer's Reference Manual (PRM)

Volume 2 Part 2: 3D/Media – Media (SandyBridge)

For the 2011 Intel Core Processor Family

May 2011

Revision 1.0

NOTICE:

This document contains information on products in the design phase of development, and Intel reserves the right to add or remove product features at any time, with or without changes to this open source documentation.

Doc Ref #: IHD-OS-V2 Pt2 – 05 11

Creative Commons License

You are free to Share — to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any
way that suggests that they endorse you or your use of the work).

No Derivative Works. You may not alter, transform, or build upon this work.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The SandyBridge chipset family, Havendale/Auburndale chipset family, Intel® 965 Express Chipset Family, Intel® G35 Express Chipset, and Intel® 965GMx Chipset Mobile Family Graphics Controller may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel® sales office or your distributor to obtain the latest specifications and before placing your product order. I2C is a two-wire communications bus/protocol developed by Philips. SMBus is a subset of the I2C bus/protocol and was developed by Intel®. Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

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

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

Copyright © 2011, Intel Corporation. All rights reserved.

Contents

1. Media and General Purpose Pipeline 4

1.1 Introduction 4
1.1.1 Hardware Feature Map in Products 4
1.2 Media Pipeline Overview 5
1.3 Programming Media Pipeline 7
1.3.1 Command Sequence 7
1.3.2 Interrupt Latency 10
1.4 Video Front End Unit 11
1.4.1 Interfaces 12
1.4.2 Mode of Operations 13
1.4.3 Parameterized Media Walker [DevSNB+] 14
1.5 Thread Spawner Unit 23
1.5.1 Basic Functions 24
1.5.2 Interfaces 29
1.6 Media State Model [DevSNB+] 30
1.7 Media State and Primitive Commands 30
1.7.1 MEDIA_VFE_STATE Command [DevSNB+] 30
1.7.2 MEDIA_CURBE_LOAD Command [DevSNB+] 33
1.7.3 MEDIA_INTERFACE_DESCRIPTOR_LOAD Command [DevSNB+] 34
1.7.4 INTERFACE_DESCRIPTOR_DATA [DevSNB+] 35
1.7.5 MEDIA_GATEWAY_STATE Command [DevSNB] 38
1.7.6 MEDIA_STATE_FLUSH Command [DevSNB] 40
1.7.7 MEDIA_OBJECT Command [DevSNB+] 41
1.7.8 MEDIA_OBJECT_PRT Command 76
1.7.9 MEDIA_OBJECT_WALKER Command [DevSNB+] 79
1.8 Media Messages 83
1.8.1 Thread Payload Messages 83
1.8.2 Thread Spawn Message 88

1. Media and General Purpose Pipeline

1.1 Introduction

This section covers the programming details for the media (general purpose) fixed function pipeline. The **media pipeline** is positioned in parallel with the 3D fixed function pipeline. It is so named as its initial (and primary) usage is to provide media functionalities and it does have media specific fixed function capability. However, the fixed functions are designed to have the general capability of controlling the shared functions and resources, feeding generic threads to the Execution Units to be executed, and interacting with such generic threads during run time. The media pipeline can be used for non-media applications, and therefore, can also be referred to as the **general purpose pipeline**. *For the rest of this chapter, we will refer this fixed function pipeline as the media pipeline, keeping in mind its general purpose capability.*

Concurrency of the media pipeline and the 3D pipeline is not supported. In other words, only one pipeline can be activated at a given time. Switching between the two pipelines within a single context is supported using the MI_PIPELINE_SELECT command.

The followings are some media application examples that can be mapped onto the media pipeline. All these applications are functional; however, what level of performance can be achieved depends on the hardware configuration and is beyond the scope of this document.

- MPEG-2 decode acceleration with HWMC
- MPEG-2 decode acceleration with IS/IDCT and forward
- MPEG-2 decode acceleration with VLD and forward
- WMV-9 decode acceleration with post filters
- WMV-9 decode acceleration with HWMC and post filters
- WMV-9 decode acceleration with IS/IDCT and forward
- AVC decode acceleration with HWMC and forward including Loop Filter
- VC1 decode acceleration with HWMC and forward including Loop Filter
- Advanced deinterlace filter (motion detected or motion compensated deinterlace filter)
- Video encode acceleration (with various level of hardware assistant)

1.1.1 Hardware Feature Map in Products

The following table lists the hardware features in the media pipe.

Table 1-1. Video Front End Features in Device Hardware

Features/ Device	[DevGT+]
Generic Mode	**Y**
Root Threads	**Y**
Parent/Child Threads	**Y**
SRT (Synchronized Root Threads)	**Y**
PRT (Persistent Root Thread)	**Y**
Interface Descriptor Remapping	N
Interface Descriptor Remapping	N
IS Mode (HW Inverse Scan)	N
VLD Mode (HW MPEG2 VLD)	N
AVC MC Mode	N
AVC IT Mode (HW AVC IT)	N
AVC ILDB Filter (in Data Port)	N
VC1 MC Mode	N
VC1 IT Mode (HW VC1 IT)	N
Stalling HW Scoreboard	**Y**
Non-stalling HW Scoreboard	**Y**
HW Walker	**Y**
HW Timer	**Y**
Pipelined State Flush	**Y**
HW Barrier	**Y**

1.2 Media Pipeline Overview

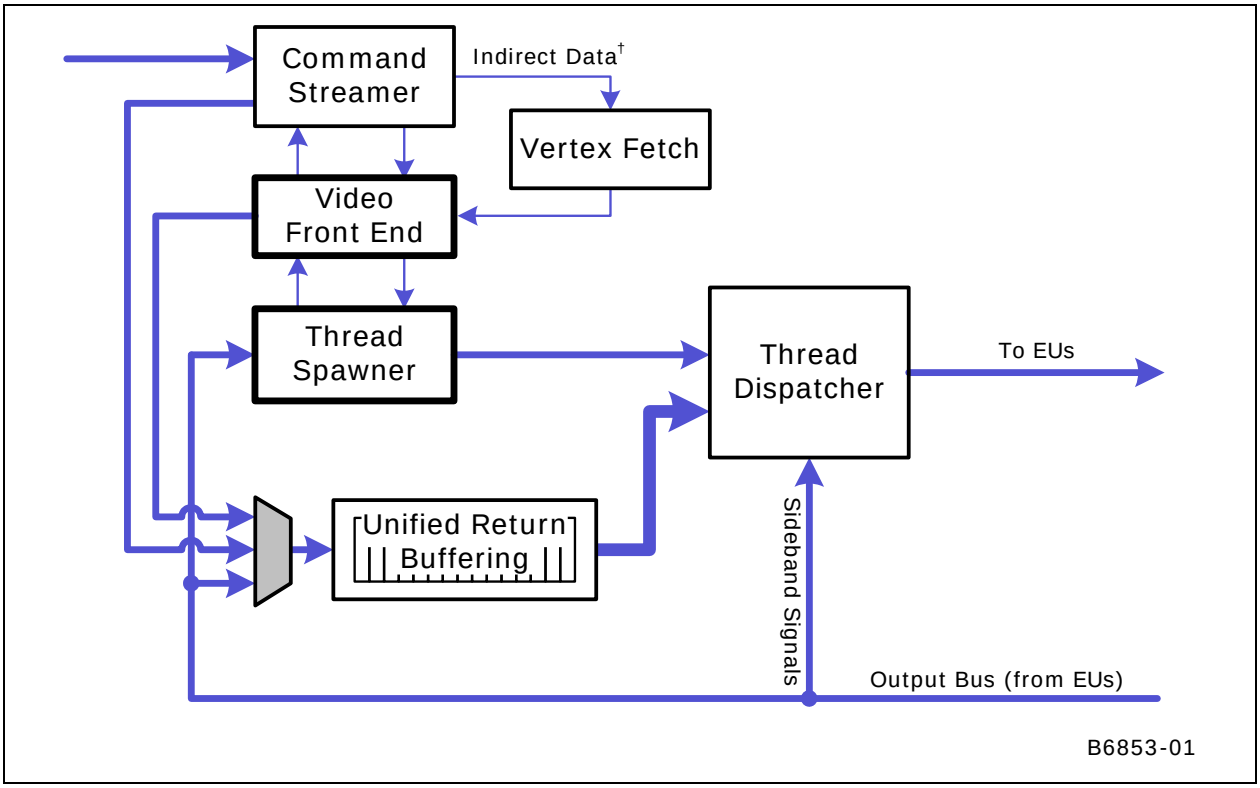
The media (general purpose) pipeline consists of two fixed function units: Video Front End (VFE) unit and Thread Spawner (TS) unit. VFE unit interfaces with the Command Streamer (CS), writes thread payload data into the Unified Return Buffer (URB) and prepares threads to be dispatched through TS unit. VFE unit also contains a hardware Variable Length Decode (VLD) engine for MPEG-2 video decode. TS unit is the only unit of the media pipeline that interfaces to the Thread Dispatcher (TD) unit for new thread generation. It is responsible of spawning root threads (short for the root-node parent threads) originated from VFE unit and spawning child threads (can be either a leaf-node child thread or a branch-node parent thread) originated from the Execution Units (EU) by a parent thread (can be a root-node or a branch-node parent thread).

The fixed functions, VFE and TS, in the media pipeline, in most cases, share the same basic building blocks as the fixed functions in the 3D pipeline. However, there are some unique features in media fixed functions as highlighted by the followings.

- VFE manages URB and only has write access to URB; TS does not interface to URB.

- When URB Constant Buffer is enabled, VFE forwards TS the URB Handler for the URB Constant Buffer received from CS.
- TS interfaces to TD; VFE does not.
- TS can have a message directed to it like other shared functions (and thus TS has a shared function ID), and it does not snoop the Output Bus as some other fixed functions in the 3D pipeline do.
- A root thread generated by the media pipeline can only have up to one URB return handle.
- If a root thread has a URB return handle, VFE creates the URB handle for the payload to initiating the root thread and also passes it alone to the root thread as the return handle. The root thread then uses the same URB handle for child thread generation.
- If URB Constant Buffer is enabled and an interface descriptor indicates that it is also used for the kernel, TS requests TD to load constant data directly to the thread's register space. For root thread, constant data are loaded after R0 and before the data from the other URB handle. For child thread, as the R0 header is provided by the parent thread, Thread Spawner splits the URB handles from the parent thread into two and inserts the constant data after the R0 header.
- A root thread must terminate with a message to TS. A child thread should also terminate with a message to TS.
- High streaming performance of indirect media object load is achieved by utilizing the large vertex cache available in the Vertex Fetch unit (of the 3D pipeline).

Figure 1-1. Top level block diagram of the Media Pipeline



1.3 Programming Media Pipeline

1.3.1 Command Sequence

Media pipeline uses a simple programming model. Unlike the 3D pipeline, it does not support pipelined state changes. Any state change requires an MI_FLUSH or PIPE_CONTROL command. When programming the media pipeline, it should be cautious to not use the pipelining capability of the commands described in the Graphics Processing Engine chapter.

To emphasize the non-pipeline nature of the media pipeline programming model, the programmer should note that if any one command is issued in the “Primitive Command” step, none of the state commands described in the previous steps cannot be issued without preceding with a MI_FLUSH or PIPE_CONTROL command.

Note for [DevSNB+]: With the addition of MEDIA_STATE_FLUSH command, pipelined state changes are allowed on the media pipeline. The MEDIA_STATE_FLUSH serves as a fence for state change by flushing the VFE/TS front ends but not waiting for threads to retire.

The basic steps in programming the media pipeline are listed below. Some of the steps are optional; however, the order must be followed strictly. Some usage restrictions are highlighted for illustration purpose. For details, reader should refer to the respective chapters for these commands.

1.3.1.1 Command Sequence [DevSNB+]

For [DevSNB+], the media pipeline is further simplified with fixed functions like MPEG2 VLD and AVC/VC1 IT removed. The addition includes (1) CURBE command is now unique to the media pipeline and (2) the interface descriptors are delivered directly as a media state command instead of being loaded through indirect state.

The programming model is listed as the following.

- Step1: MI_FLUSH/PIPE_CONTROL
    - This step is mandatory.
    - Multiple such commands in step 1 are allowed, but not recommended for performance reason.
- Step2: State command PIPELINE_ SELECT
    - This step is optional. This command can be omitted if it is known that within the same context media pipeline was selected before Step 1.
    - Multiple such commands in step 2 are allowed, but not recommended for performance reason.
- Step3: State commands configuring pipeline states
    - STATE_BASE_ADDRESS
        - This command is mandatory for this step (i.e. at least one).
        - Multiple such commands in this step are allowed. The last one overwrites previous ones.
        - This command must precede any other state commands below.
        - Particularly, the fields **Indirect Object Base Address** and **Indirect Object Access Upper Bound** are used to control indirect Media object load in VF.
        - The fields **Dynamics Base Address** and **Dynamics Base Access Upper Bound** are used to control indirect Curbe and Interface Descriptor object load in VF.
        - *Note: This command may be inserted before (and after) any commands listed in the previous steps (Step 1 to 3). For example, this command may be placed in the ring buffer while the others are put in a batch buffer.*

- STATE_SIP
    - This command is optional for this step. It is only required when SIP is used by the kernels.
- MEDIA_VFE_STATE
    - This command is mandatory for this step (i.e. at least one).
    - This command cause destruction of all outstanding URB handles in the system. A new set of URB handles will be generated based on state parameters, no. of URB and URB length, programmed in VFE FF state.
    - Multiple such commands in this step are allowed. The last one overwrites previous ones.
- MEDIA_CURBE_LOAD
    - This command is optional.
    - Multiple such commands in this step are allowed. The last one overwrites previous ones.
- MEDIA_INTERFACE_DESCRIPTOR_LOAD
    - This command is mandatory for this step (i.e. at least one).
    - Multiple such commands in this step are allowed. The last one overwrites previous ones.

- Step4: Primitive commands
    - MEDIA_OBJECT
        - This step is optional, but it doesn't make practical sense not issuing media primitive commands after being through previous steps to set up the media pipeline.
        - Multiple such commands in step 4 can be issued to continue processing media primitives.

With the addition of MEDIA_STATE_FLUSH command, pipelined state changes are allowed on the media pipeline. In order to support context switch for barrier groups, watermark and barrier dependencies are added to the MEDIA_STATE_FLUSH command. The usage of barrier group may have strict restriction that all threads belonging to a barrier group must all be present in order to avoid deadlock during context switch. Here are the example programming sequences to allow context switch. Note that the use of MEDIA_OBJECT_PRT is optional. It is required to support fast preemption. Similarly, MI_ARB_ON_OFF is optional and is used to support fast preemption.

- MEDIA_VFE_STATE
- MEDIA_INTERFACE_DESCRIPTOR_LOAD

- MEDIA_CURBE_LOAD (optional)
- MEDIA_GATEWAY_STATE (for example for barrier group 1)
- MEDIA_OBJECT_PRT (with VFE_STATE_FLUSH set and PRT NEEDED set.)
- MEDIA_STATE_FLUSH (with watermark set for group 1)
- MI_ARB_ON_OFF (OFF) // Arbitration must be turned off while sending objects for group 1
- Several MEDIA_OBJECT command (for barrier group 1)
- MI_ARB_ON_OFF (ON) // Arbitration is allowed
- MEDIA_STATE_FLUSH (optional, only if barrier dependency is needed)
- MEDIA_INTERFACE_DESCRIPTOR_LOAD (optional)
- MEDIA_CURBE_LOAD (optional)
- MEDIA_GATEWAY_STATE (for example for barrier group 2)
- MEDIA_STATE_FLUSH (with watermark set for group 1)
- MI_ARB_ON_OFF (OFF) // Arbitration must be turned off while sending objects for group 2
- Several MEDIA_OBJECT command (for barrier group 2)
- MI_ARB_ON_OFF (ON) // Arbitration is allowed
- …
- MI_FLUSH

1.3.2 Interrupt Latency

Command Streamer is capable of context switching between primitive commands.

For all independent threads, it is not much a problem. The interrupt latency is dictated by the longest command that is likely to have the largest number of threads. For VLD mode, such a command may be corresponding to a largest slice in a high definition video frame. This is application dependent, there are not much host software can do. For Generic mode, programmer should consider to constrain the compute workload size of each thread.

In modes with child threads, a root thread may be persist in the system for long period of time – staying until its child threads are all created and terminated. Therefore, the corresponding primitive command may also last for long time. Software designer should partition the workload to restrict the duration of each root thread. For example, this may be achieved by partitioning a video frame and assigning separate primitive commands for different data partitions.

In modes with synchronized root threads, a synchronized root thread is dependent on a previous root or child thread. This means context switch is not allowed between the primitive command for the synchronized root thread and the one for the depending thread. So no command queue arbitration should

be allowed between them. Software designer should also restrict the duration of such non-interruptible primitive command segments.

1.4 Video Front End Unit

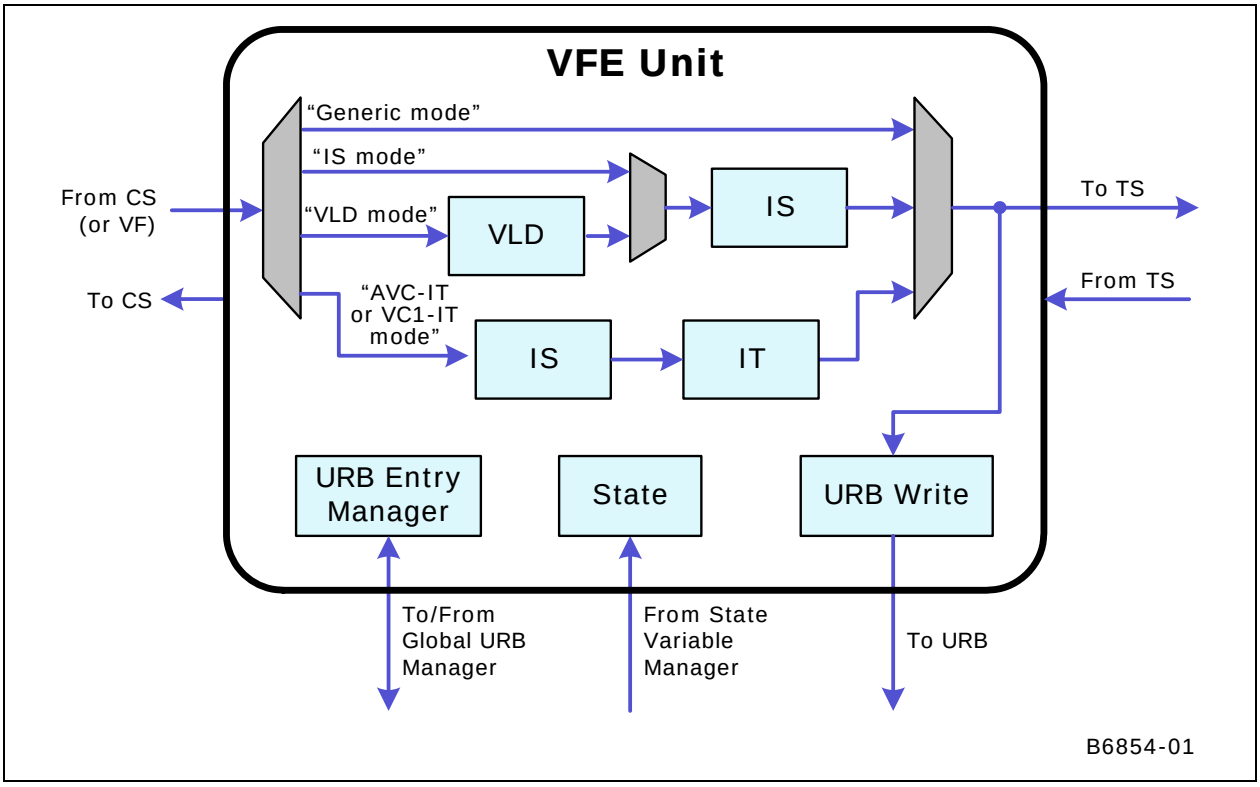
The Video Front End unit is the first fixed function unit in the media pipeline. It processes MEDIA_OBJECT commands to generate root threads by preparing the control (including interface descriptor pointers) and payload (data pushed into the GRF) for the root threads.

VFE supports three modes of operation: Generic mode, Inverse Scan mode and VLD mode.

- **Generic mode**: In the Generic mode, VFE serves as a conduit for general-purpose kernels fully configured by the host software. There is no application specific hardware enabled in this mode.

The following figure illustrates the three modes of operation. The details can be found in the rest of the sections.

Figure 1-2. VFE Functional Blocks and Modes of Operations



MEDIA_STATE_POINTERS command configures VFE in one of the three modes using. Mode switching requires media pipeline state change.

1.4.1 Interfaces

VFE unit acquires its states from Sate Variable Manager, accesses URB handles from the Global URB Manager, receives state and primitive commands from CS unit, writes thread payloads to URB, and sends new thread to TS unit. It does not directly interface to Thread Dispatcher. When VFE is ready for a thread, it sends the interface descriptor pointer for the thread to TS.

1.4.1.1 Interface to Command Streamer

VFE interfaces to CS to acquire the control data, inline data and indirect data of MEDIA_OBJECT commands. The interface supports the throughput of a given mode of operation of VFE. For example, in VLD mode and IS mode, VFE consumes one dword at a time, one dword to the variable length decoder or one dword to the inverse-scan operator. In Generic mode, VFE is capable of a much higher throughput to push indirect data (as thread payload data) into URB. As throughput for indirect data is much higher than that of inline data, when large amount of user data need to be passed through VFE unit, if applicable, it is encouraged to use indirect object load.

1.4.1.2 Interface to Thread Spawner

When a new root thread is fully assembled by VFE, VFE passes to TS the interface descriptor pointer, the URB handle information, etc. In response to this, TS processes the thread information and sends a thread request to TD.

VFE also transmits scratch memory base address received from State Variable Manager to TS, and passes on the Constant URB handle received from CS.

VFE receives URB handle dereference signal from TS.

1.4.1.3 Interface to State Variable Manager

State Variable Manager is responsible of fetching media state structure from memory. VFE only acquires its state variable upon the first primitive command. Therefore, host software is allowed to change media states before issuing primitive commands. As media pipeline does not support pipelined state change, a pipeline flush is required before any state change to make sure that there are no outstanding primitive commands in the pipeline.

1.4.1.4 Interface to Global URB Manager

VFE is responsible for managing URB handles for all root threads. Upon state change, VFE allocates URB handles through the Global URB Manager. VFE manages the URB handles in a circular buffer. URB handle referencing is in a strict order (taking from the head of the circular buffer), even though the handle dereferencing may occur out of order.

When starting a root thread, VFE reference one and only one URB handle, forwarding it to TS. TS then forwards this handle to TD for thread dispatching.

The URB handle for a root thread is used in two ways: (1) serving as buffer space for VFE to assemble thread payload, and (2) serving as the return URB buffer for the root thread to assemble child threads and their payload.

TS sends an indication to VFE when it is safe to dereference the URB handle, and VFE dereferences it. After a URB handle has been dereferenced, VFE can assign it to a new thread.

1.4.1.5 Interface to URB

VFE sends the assembled root thread payload to URB via a wide data bus. In Generic mode, the data comes from the command as inline or indirect data objects. In IS mode, the inline data is directly assembled as URB register wide payloads, and the indirect data are assembled through the Inverse Scan logic. In VLD mode, the data is decoded from the indirect object (i.e. bitstream data).

1.4.2 Mode of Operations

1.4.2.1 Generic Mode

In the Generic mode, VFE serves as a conduit for general-purpose kernels fully configured by the host software. As there is no special fixed function logic used, the Generic mode can also be viewed as a 'pass-through' mode. In this mode, VFE generates a new thread for each MEDIA_OBJECT command. The payload contained in the MEDIA_OBJECT command (inline and/or indirect) is streamed into URB. The interface descriptor pointer is computed by VFE based on the interface descriptor offset value and the interface descriptor base pointer stored in the VFE state. VFE then forwards the interface descriptor pointer and the URB handle to TS to generate a new root thread. Many media processing applications can be supported using the Generic mode: MPEG-2 HWMC, frame rate conversion, advanced deinterface filter, to name a few.

1.4.2.1.1 Interface Descriptor Selection

After populating the URB with the data, VFE notifies TS to initiate the thread. TS needs an interface descriptor pointer to fetch the information for thread initiation. A list of interface descriptors is arranged by the host software as a descriptor array in memory, as shown in the media state model in **Error! Reference source not found.**.

VFE obtains the interface descriptor base pointer from the VFE state structure. The offset into the list of interface descriptors comes from MEDIA_OBJECT command. Each interface descriptor has a fixed size. VFE uses a multiple of the fixed size and the offset to add to the base pointer, and creates the final interface descriptor pointer to be sent to TS.

TS fetches the interface descriptor through the Instruction State Cache (ISC) using the interface descriptor pointer. TS then initializes the thread through the Thread Dispatcher. The interface descriptor pointer is given to TS by VFE for a root thread and by a thread for a child thread. The R0 header is formed by TS for a root thread and is stored in URB by the parent thread for a child thread.

1.4.2.1.2 Scratch Space Allocation

TS handles the allocation of scratch space. Since TS does not have a normal state interface, VFE receives the scratch space configuration with the VFE state, then forwards the configuration to TS with the interface descriptor pointer.

1.4.3 Parameterized Media Walker [DevSNB+]

The Parameterized Media Walker is a hardware thread generation mechanism that creates threads associated with units in a generalized 2-dimensional space, for example, blocks in a 2D image. With a small number of unit step vectors, the walker can implement a large number of walking patterns as described hereafter. This command may provide functions that are normally handled by the host software, thus, may be used to simplify the host software and GPU interface.

The walker described herein is doubly nested, where essentially a "local" walker can perform a variety of 2-dimensional walking patterns and a "global" walker can perform similar 2-dimensional walking patterns upon many local walkers. The local walker has 3 levels (outer, middle, and inner) while the global walker has 2 levels (outer and inner). Thus, the algorithm has 5-nested loops that modify local state based on user-defined unit step vectors.

The Walker's programmability is derived from:

- The walker traverses a unit-normalized surface. Some example unit sizes:
    - 1x1: Walking pixels
    - 4x4: Walking sub-blocks
    - 16x16: Walking macro-blocks
    - 32x16: Walking macro-block-pairs
- The use of unit step vectors to describe the motion at each of level of nesting
- Starting locations for the local and global walkers
- Block sizes of the local and global walker
- And a small number of special mode controls for the inner-most loop which are aimed at efficiently dividing an image into two balanced workloads for dual-slice designs.

1.4.3.1 Walker Parameter Description

The global and local loops are both described by the same four parameters:

- Resolution,
- Starting location,
- Outer unit vector,
- Inner unit vector

The local inner loop has some special modes that will be described later. A table of the user inputs and some example values are given below:

GLOBAL LOOP PARAMETERS							
Global Resolution		Global Start		Outer Loop Unit Vector		Inner Loop Unit Vector	
X	Y	X	Y	X	Y	X	Y
120	68	0	0	32	0	0	32

LOCAL LOOP PARAMETERS							
Block Resolution		Local Start		Outer Loop Unit Vector		Inner Loop Unit Vector	
X	Y	X	Y	X	Y	X	Y
32	32	0	0	1	0	-2	2

LOCAL INNER LOOP SPECIAL MODE SELECTS							
Dual Mode	Repel	Attract			ExtraSteps	X	Y
TRUE	FALSE	FALSE			1	0	1

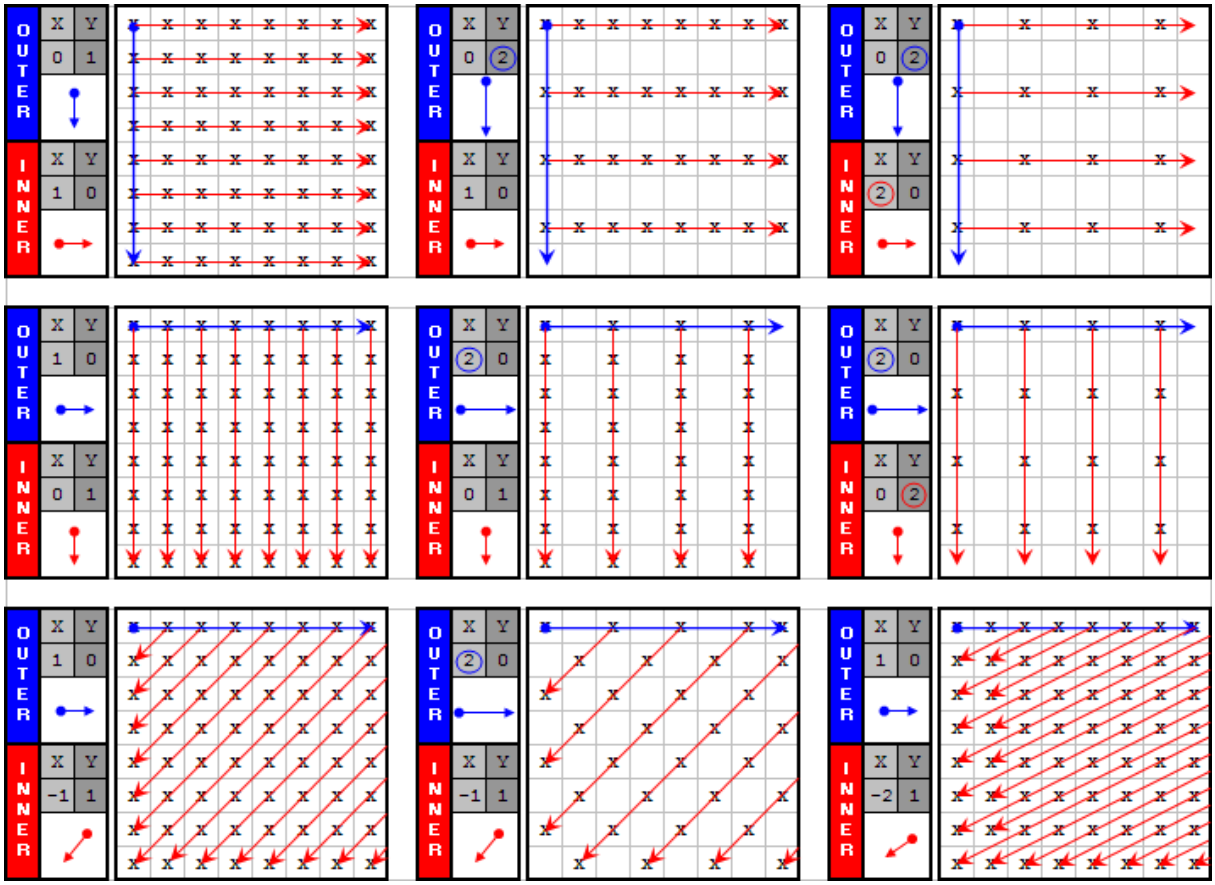
It should be emphasized that the value of what a “unit” represents is implicitly defined by the user. In other words, the walker traverses a “unit normalized space” that is not inherently bound to pixel walking. If the smallest unit of work the user wants to walk is a 4x3 block of pixels, you can program the inner loop to step (4,3) or (1,1):

- In the first case (4,3) the user is walking in units of pixels
- In the second case (1,1) the user is walking in units of 4x3 blocks of pixels.

It should be noted that hardware doesn’t contain enough bits for pixel walking for pixel resolution like 1920x1088. The intended usage of the walker is for block walking whereas the block size is not relevant to the walker parameters.

1.4.3.2 Basic Parameters for the Local Loop

The local inner and outer loop xy-pair parameters alone can describe a large variety of primitive walking patterns. Below are 9 primitive walking patterns generated by varying only the inner and outer unit step vectors of the local loop:

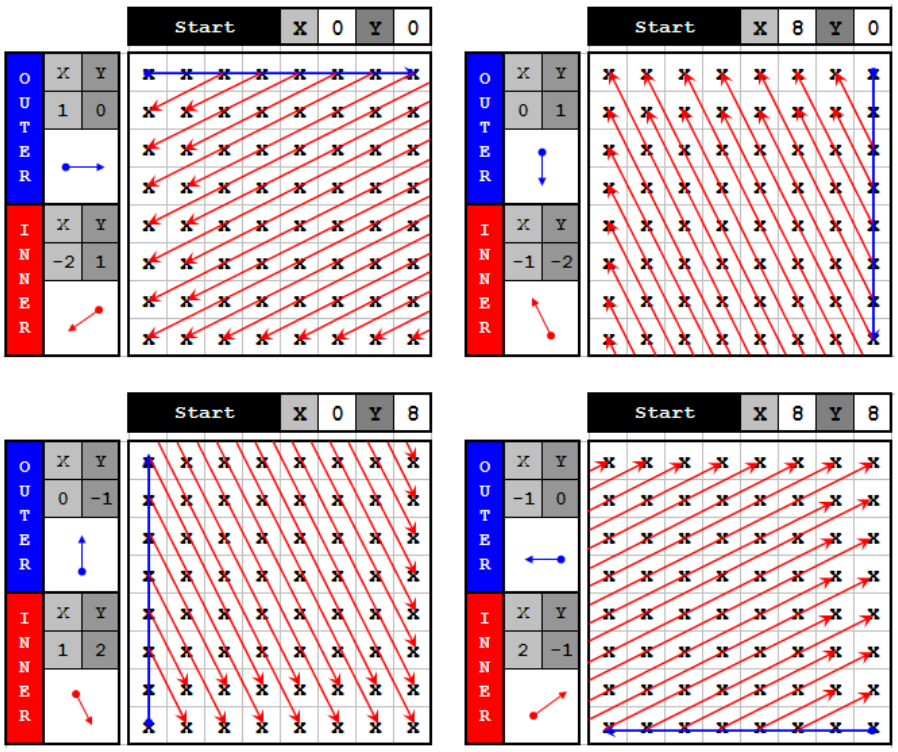


- The top row shows the outer unit vector pointing down (+Y) and the inner unit vector pointing right (+X). Rows and columns can easily be skipped by increasing the unit step vectors above one.

- The middle row the outer unit vector pointing right (+X) and the inner unit vector pointing down (+Y). Again, rows and columns are skipped by increasing the unit step vectors beyond one.

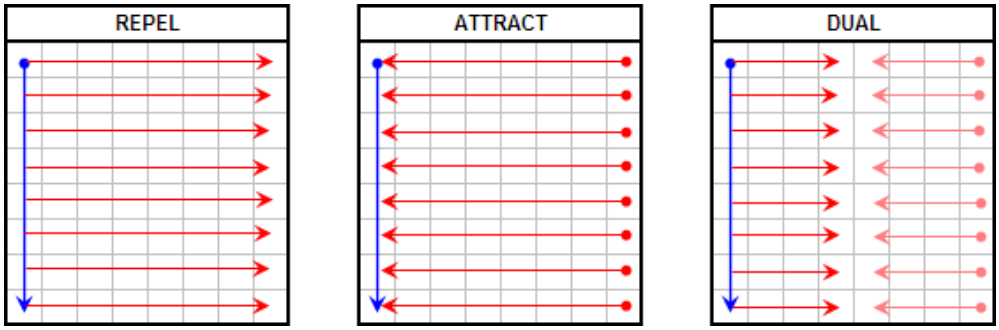
- The last row shows the capability to walk angles not perpendicular to the edge. The 1st shows a 45º walking pattern by setting the inner unit vector to (-1,1). The 2nd shows a checkerboard pattern by skipping every other outer loop and retaining the inner unit vector of (-1,1). The 3rd shows a 26.5º walking pattern by setting the inner unit vector to (-2,1).

The block resolution, shown as [8,8], and the starting location, currently [0,0], can be varied and the above patterns can be stretched and rotated many ways. The diagram below shows an example of where the start position and unit step vectors can be set to achieve a full rotation of the same pattern:

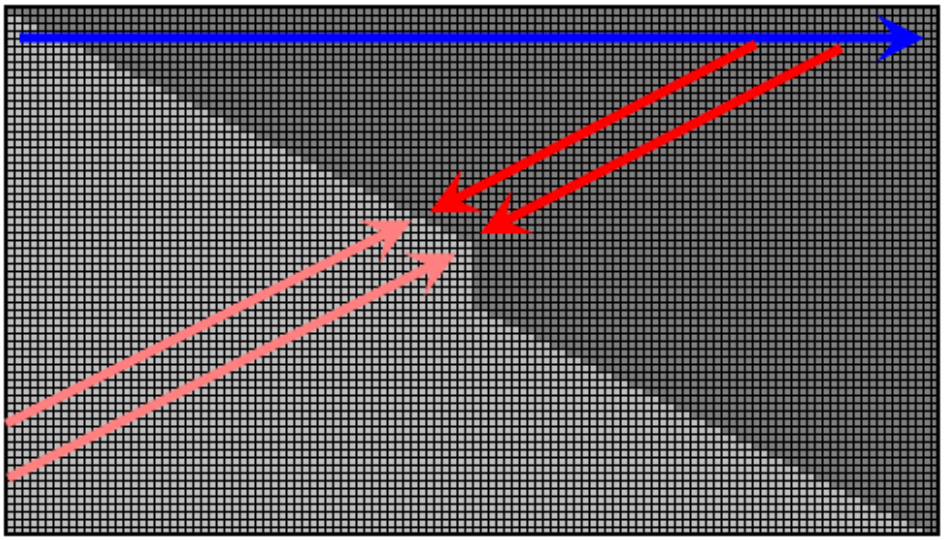


1.4.3.3 Dual Mode of Local Loop

The local Inner Loop Special mode selects are included to aid in the distribution of work, specifically with two slices in mind. Essentially, the local inner loop can be bisected and each half-walk can be directed inward towards the center of the image (dual). The local inner loop need not be bisected, and can either move away from the outer loop (repel) or move towards it (attract) when an even split is not desired:

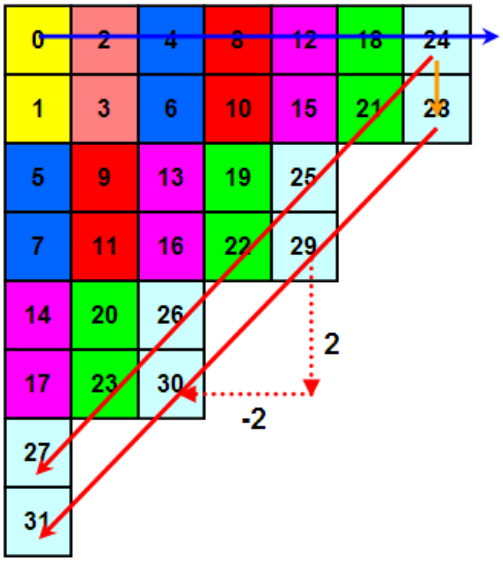


In Dual mode, the sequence will alternate between two half-walks such that every-other output would go to the same slice. This effect will produce a more balanced workload to two slices as shown in the example below where the color of the block represents which slice it was dispatched to. This is the walker's approach to fine-grained parallelism.



1.4.3.4 MbAff-Like Special Case in Local Loop

The local loop has an additional middle loop that is used to achieve some specific walking patterns, with MBAFF mode especially in mind. A pattern to handle MBAFF AVC content is to walk the top macroblocks of all macroblock pairs (MB-pairs) on a wavefront followed by the respective bottom macroblocks. The pattern is shown below.

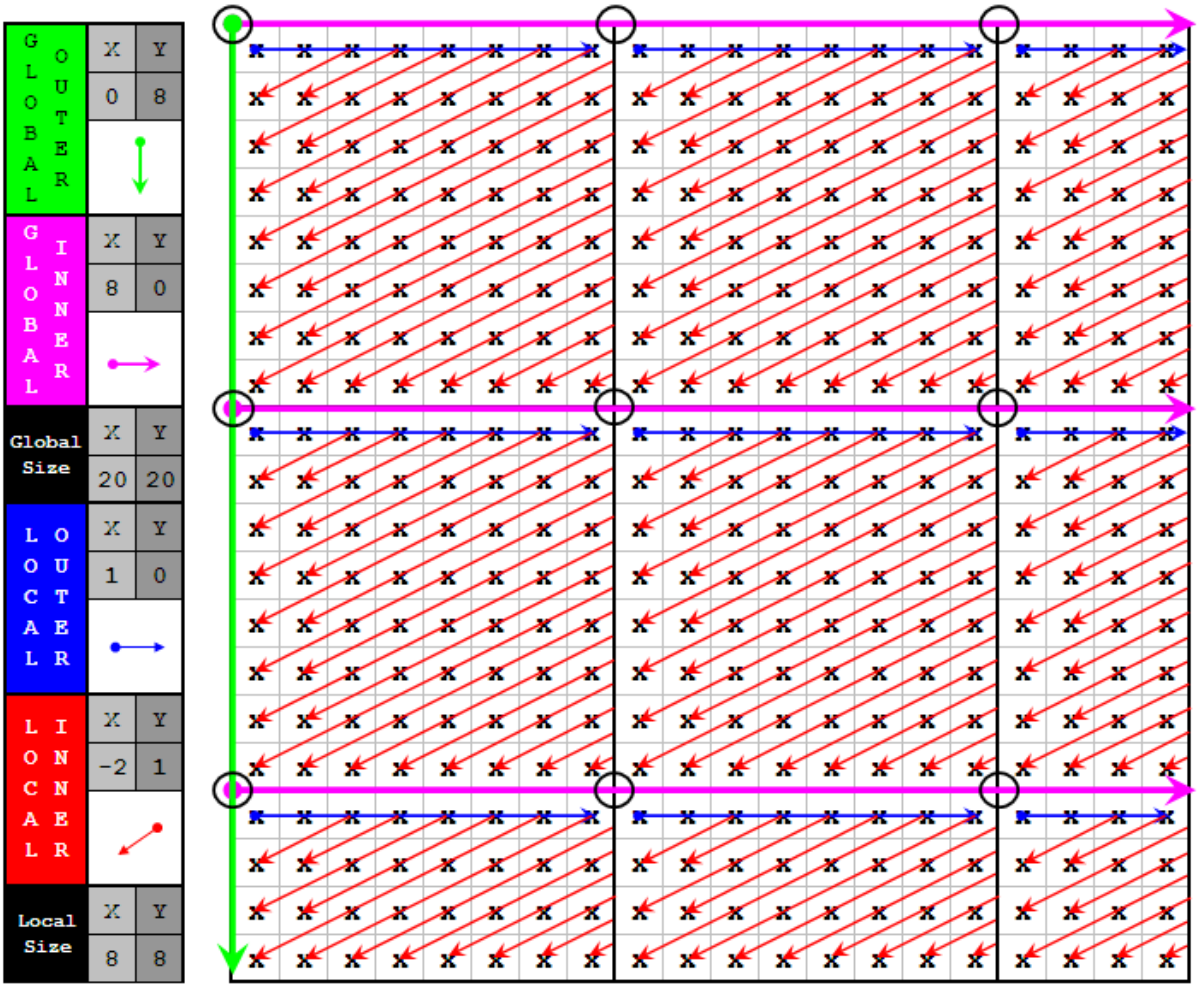


The outer loop unit step vector would be [1, 0] and the inner loop unit step vector would be [-2, 2]. A third loop is necessary to repeat the inner loop, only shifted down a unit before restarting. Thus, a middle loop with a unit step vector of [0,1] would achieve this MBAFF pattern. Additionally, the number of “extra steps” taken by the middle loop would be 1 in this case.

The addition of a middle loop also creates more overall flexibility, which seems necessary due to the integer-based unit step vector solution proposed (Manhattan distance issues etc.).

1.4.3.5 Global Loop

The same set of general parameters is used to describe the global loop as well. Thus, a global loop that is walking a raster-scan pattern can be combined with a local loop that is walking a 26.5º pattern (or vice-versa). As shown in the example below, if the local block size ([8,8]) is not an even multiple of the global resolution ([20,20]), the slack is still processed by dynamically changing the local block resolution.



The global loop will always resolve to be the upper-left corner of the local loop, shown above black circles. Note that local loop can still start in any corner of the local block, but the local (0,0) will always be the location where global loop begins the local loop, hence the upper-left corner.

The user can specify where the staring location of the global loop as with the local loop. If the user were to set the global starting location to (16,16) in the previous example, after inverting the global outer and global inner unit step vectors the same pattern would be achieved in the reverse order. Note that the slack would still be handled along the right and bottom edge of the global image in that case. The user could have also started at (12,12) in which case the slack would be handled on the left and top faces.

1.4.3.6 Walker Algorithm Description

The walker algorithm has been tested and optimized in software. A high-level pseudo-code description is given below:

```
Walker(){ //C-Style Pseudo-Code of Walker Algorithm
    Load_Inputs_And_Initialize();
    While (Global_Outer_Loop_In_Bounds()){
        Global_Inner_Loop_Intialization();
        While (Global_Inner_Loop_In_Bounds()){
            Local_Block_Boundary_Adjustment();
            Local_Outer_Loop_Initialization();
            While (Local_Outer_Loop_In_Bounds()){
                Local_Middle_Loop_Initialization();
                While (Local_Middle_Steps_Remaining()){
                    Local_Inner_Loop_Initialization();
                    While (Local_Inner_Loop_Is_Shrinking()){
                        Execute();
                        Calculate_Next_Local_Inner_X_Y();
                    } //End Local Inner Loop
                    Calculate_Next_Local_Middle_X_Y();
                } //End Local Middle Loop
                Calculate_Next_Local_Outer_X_Y();
                Calculate_Next_Local_Inverse_Outer_X_Y();
            } //End Local Outer Loop
            Calculate_Next_Global_Inner_X_Y();
        } //End Global Inner Loop
        Calculate_Next_Global_Outer_X_Y();
    } //End Global Outer Loop
} //End Walker
```

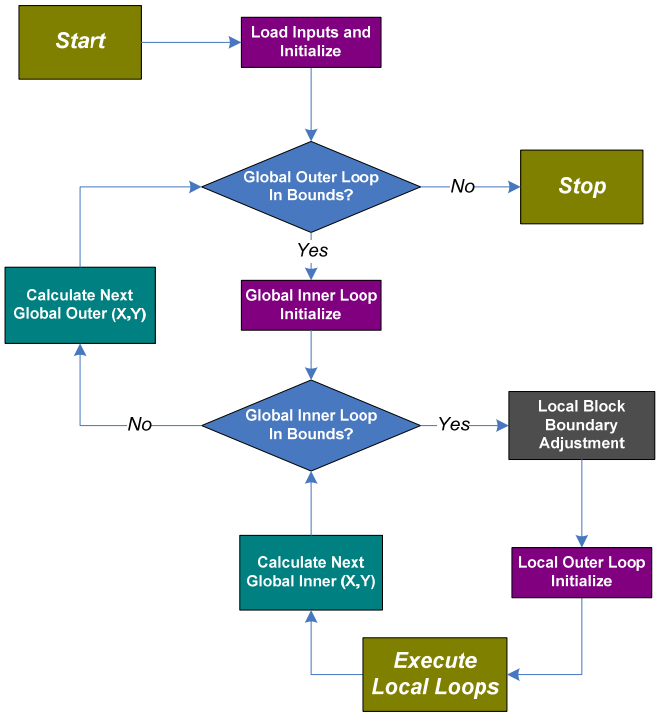
The pseudo-code has the following characteristics:

- There are 5 levels of iteration
- The highest 2 levels are called “global” and the lowest 3 levels are called “local”
    - The global loop is split into an outer and an inner loop.
    - The local loop is split into an outer, a middle, and an inner loop.
    - A bounding box for the global and local resolution is defined by the user.
    - The starting location within each bounding box is also specified by the user.
- Each of the 5 loops has its own persistent
    - Current position (x,y)
    - Unit step vector (x,y)
- The final output (x,y) is a summation of the global x,y and the local x,y.
- The next (x,y) for given level can be calculated while the next lower level is still executing. Additionally, the result can be used to check to see if the current level will execute again once control is returned.

The flow of the global outer and inner loops is:

1. Check a bound condition
2. Initialize the next level loop
3. Execute the next level loop
4. When the next level loop fails its condition, calculate the next position for the current loop level and repeat.

Figure 1-3. Walker algorithm flowchart for the Global Loop

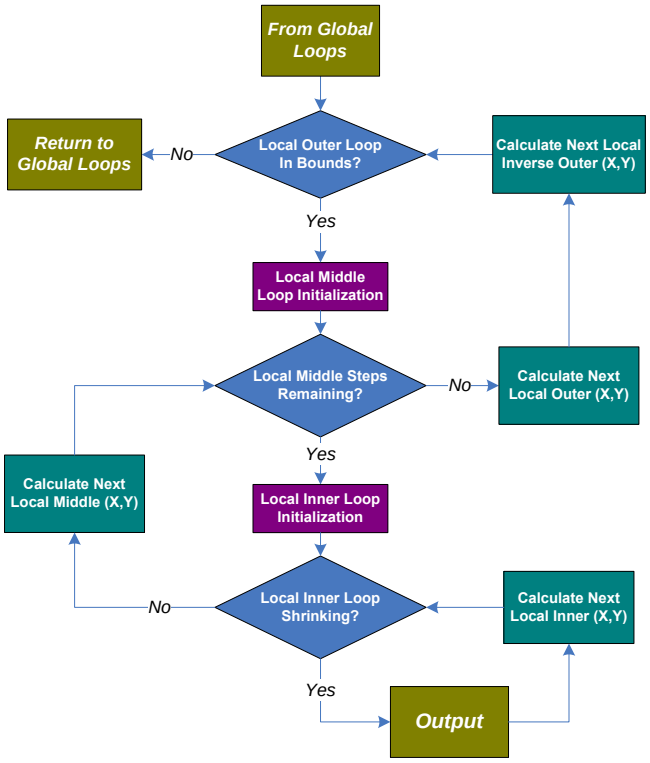


Take note of the grey box "Local Block Boundary Adjustment". This logic is necessary to adjust the local block size when the distance between the current global position to the edge of the image is less than the local resolution. Additionally, the local starting positions might be modified here as well if the defined starting position is larger than the new local block size.

The flow of the 3 local loops does not vary much from the 2 global loops. The differences are:

- In addition to a boundary check, the local middle loop also ensures the number of middle steps is less than or equal to the user defined "number of extra steps".
- The local inner loop only checks to see if the prior distance between the x,y starting and ending points are greater than their current distance. If this is true, it implies that the two inner loops are converging towards each other.
- When the middle loop check fails, both the starting points (local outer) and ending points (local inner) are updated.

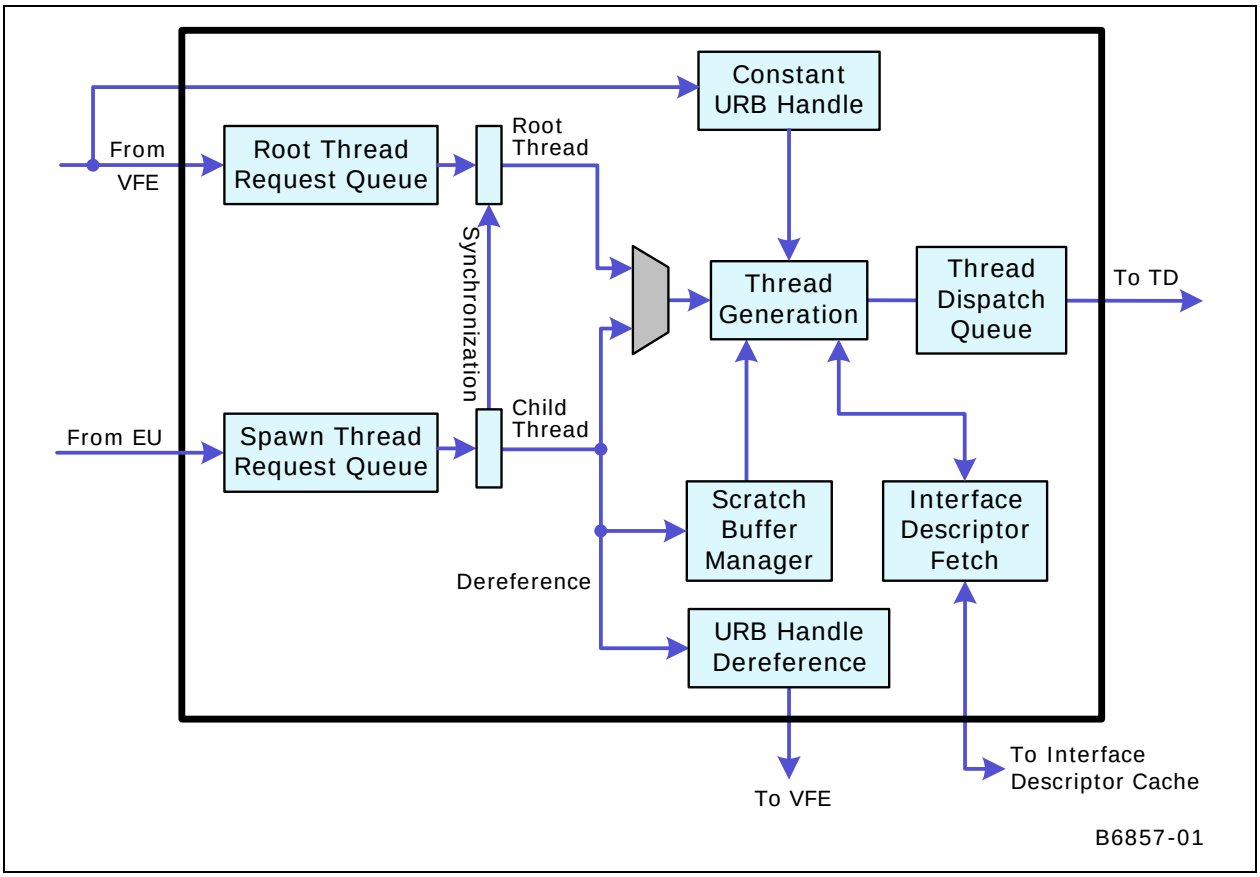
Figure 1-4. Walker algorithm flowchart for the Local Loop



1.5 Thread Spawner Unit

The Thread Spawner (TS) unit is responsible for making thread requests (root and child) to the Thread Dispatcher, managing scratch memory, maintaining outstanding root thread counts, and monitoring the termination of threads.

Figure 1-5. Thread Spawner block diagram



1.5.1 Basic Functions

1.5.1.1 Root Threads Lifecycle

Thread requests sourced from VFE are called **root threads**, since these threads may be creating subsequent (child) threads. A root thread may a macroblock thread created by VFE as in VLD mode, or may be a general-purpose thread assembled by VFE according to full description provided by host software in Generic mode.

Thread requests are stored in the Root Thread Queue. TS keeps everything needed to get the root threads ready for dispatch and then tracks dispatched threads until their retirement.

TS arbitrates between root thread and child thread. The root thread request queue is in the arbitration only if the number of outstanding threads does not exceed the maximum root thread state variable. Otherwise, the root thread request queue is stalled until some other root threads retire/terminate.

Once a root thread is selected to be dispatched, its lifecycle can be described by the following steps:

1. TS forwards the interface descriptor pointer to the L1 interface descriptor cache (a small fully associated cache containing up to 4 interface descriptors). The interface descriptor is either

found in the cache or a corresponding request is forwarded to the L2 cache. Interface descriptors return back to TS in requesting order.

- Once TS receives the interface descriptor, it checks whether maximum concurrent root thread number has reached to determine whether to make a thread dispatch request or to stall the request until some other root threads retire. If the thread requests the use of scratch memory, it also generates a pointer into the scratch space.

2. TS then builds the transparent header and the R0 header.

3. Finally, TS makes a thread request to the Thread Dispatcher.

4. TS keeps track of dispatched thread, and monitors messages from the thread (resource dereference and/or thread termination). When it receives a root thread termination message, it can recover the scratch space and thread slot allocated to it. The URB handle may also be dereferenced for a terminated root thread for future reuse. It should be noted that URB handle dereference may occur before a root thread terminates. See detailed description in the Media Message section.

   - It is the root thread's responsibility (software) to guarantee that all its children have retired before the root thread can retire.

1.5.1.2 URB Handles

VFE is in charge of allocating URB handles for root threads. One URB handle is assigned to each root thread. The handle is used for the payload into the root thread.

If Children Present is not set (root-without-child case), TS signals VFE to dereference the URB handle immediately after it receives acknowledgement from TD that the thread is dispatched.

If Children Present is set (root-with-child case), the URB handle is forwarded to the root thread and serves as the return URB handle for the root thread. TS does not signal deference at the time of dispatch. TS signals URB handle deference only when it receives a resource dereference message from the thread.

Children Present is a command variable in the _OBJECT command

1.5.1.3 Root to Child Responsibilities

Any thread created by another thread running in an EU is called a **child thread**. Child threads can create additional threads, all under the tree of a root which was requested via the VFE path.

A root thread is responsible of managing pre-allocated resources such as URB space and scratch space for its direct and indirect child threads. For example, a root thread may split its URB space into sections. It can use one section for delivering payload to one child thread as well as forwarding the section to the child thread to be used as return URB space. The child thread may further subdivide the URB section into subsections and use these subsections for its own child threads. Such process may be iterated. Similarly, a root thread may split its scratch memory space into sections and give one scratch section for one child thread.

TS unit only enforces limitation on number of outstanding root threads. It is the root threads' responsibility to limit the number of child threads in their respected trees to balance performance and avoid deadlock.

1.5.1.4 Multiple Simultaneous Roots

Multiple root threads are allowed concurrently running in GEN4 execution units. As there is only one scratch space state variable shared for all root threads, all concurrent root thread requiring scratch space share the same scratch memory size. Figure 1-6 depicts two examples of thread-thread relationship. The left graph shows one single tree structure. This tree starts with a single root thread that generates many child threads. Some child threads may create subsequent child threads. The right graph shows a case with multiple disconnected trees. It has multiple root threads, showing sibling roots of disconnected trees. Some roots may have child threads (branches and leafs) and some may not.

There is another case (as shown in Figure 1-7) where multiple trees may be connected. If a root is a synchronized root thread, it may be dependent on a preceding sibling root thread or on a child thread.

Figure 1-6. Examples of thread relationship

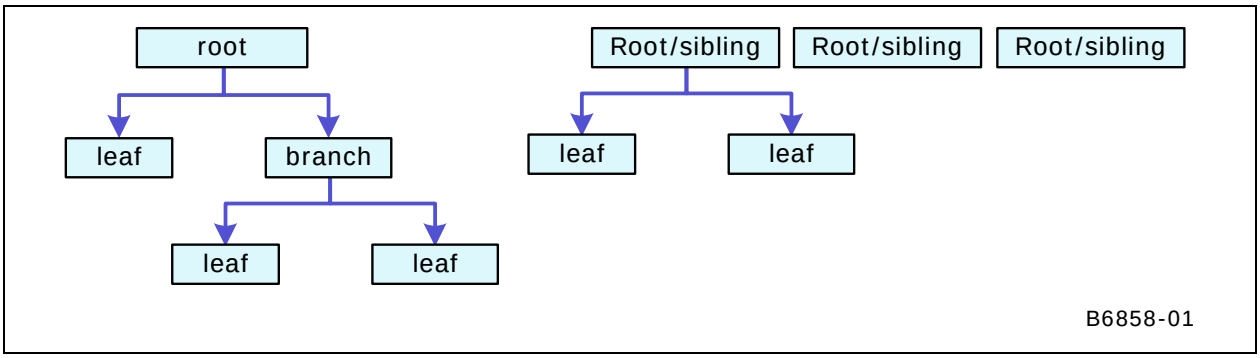
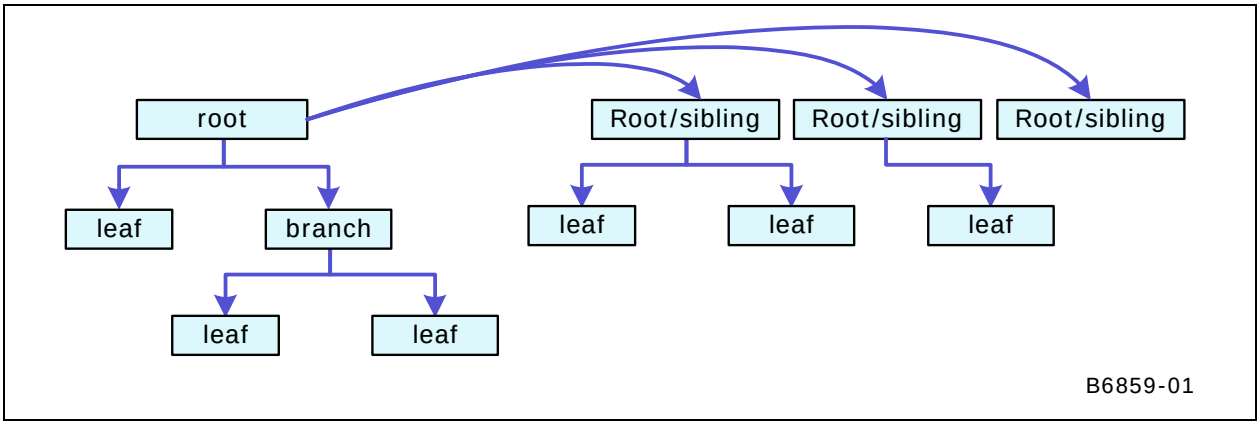


Figure 1-7. A example of thread relationship with root sibling dependency



1.5.1.5 Synchronized Root Threads

A synchronized root thread (SRT) originates from a MEDIA_OBJECT command with Thread Synchronization field set. Synchronized root threads share the same root thread request queue with the non-synchronized roots. A SRT is not automatically dispatched. Instead, it stays in the root thread request queue until a spawn-root message is at the head of the child thread request queue. Conversely, a spawn-root message in the child thread request queue will block the child thread request queue until the head of

root thread request queue is a SRT. When they are both at the head of queues, they are taken out from the queue at the same time.

A spawn-root message may be issued by a root thread or a child thread. There is no restriction. However, the number of spawn-root messages and the number of SRT must be identical between state changes. Otherwise, there can be a deadlock. Furthermore, as both requests are blocking, synchronized root threads must be used carefully to avoid deadlock.

When Scoreboard Control is enabled, the dispatch of a SRT originated from a MEDIA_OBJECT_EX command is still managed by the same way in addition to the hardware scoreboard control.

1.5.1.6 Deadlock Prevention

Root threads must control deadlock within their own child set. Each root is given a set of preallocated URB space; to prevent deadlock it must make sure that all the URB space is not allocated to intermediate children who must create more children before they can exit.

There are limits to the number of concurrent threads. The upper bound is determined by the number of execution units and the number of threads per EU. The actual upper bound on number of concurrent threads may be smaller if the GRF requirement is large. Deadlock may occur if a root or intermediate parent cannot exit until it has started its children but there is no space (for example, available thread slot in execution units) for its children to start.

To prevent deadlock, the maximum number of root threads is provided in VFE state. The Thread Spawner keeps track of how many roots have been spawned and prevents new roots if the maximum has been reached. When child threads are present, it is software's responsible of constraining child thread generation, particularly the generation of child threads that may also spawn more child threads.

Child thread dispatch queue in TS is another resource that needs to be considered in preventing deadlock. The child thread dispatch queue in TS is used for (1) message to spawn a child thread, (2) message to spawn a synchronized root thread, and (3) thread termination message. If this queue is full, it will prevent any thread to terminate, causing deadlock.

For example, if an application only has one root thread (max # of root threads is programmed to be one). This root thread spawns child threads. In order to avoid deadlock, the maximum number of outstanding child thread that this root thread can spawn is the sum of the maximum available thread slots plus the depth of the child thread dispatch queue minus one.

Max_Outstanding_Child_Threads = (Thread Slot Number – 1) + (TS Child Queue Depth – 1)

Adding other root threads (synchronized and/or non-synchronized) to the above example, the situation is more complicated. A conservative measure may have to use to prevent deadlock. For example, the root thread spawning child threads may have to exclude the max number of root threads as in the following equation to compute the maximum number of outstanding child threads to be dispatched.

Max_Outstanding_Child_Threads = (Thread Slot Number – 1) + (TS Child Queue Depth – 1) – (Max Root Threads-1)

Table 1-2. TS Resource Available in Device Hardware

Device	Child Thread Dispatch Queue Depth
[DevSNB+]	OPEN

1.5.1.7 Child Thread Lifecycle

When a (parent) thread creates a child thread, the parent thread behaves like a fixed function. It provides all necessary information to start the child thread, by assembling the payload in URB (including R0 header) and then sending a spawn thread message to TS with following data:

- An interface descriptor pointer for the child thread.
- A pointer for URB data

The interface descriptor for a child may be different from the parent – how the parent determines the child interface descriptor is up to the parent, but it must be one from the interface descriptor array on the same interface descriptor base address.

The URB pointer is not the same as a URB handle. It does not have an URB handle number and does not appear in any handle table. This is acceptable because the URB space is never reclaimed by TS after a child is dispatched, but rather when the parent releases its original handles and/or retires.

The child request is stored in the child thread queue. The depth of the queue is limited to 8, overrun is prevented by the message bus arbiter which controls the message bus. The arbiter knows the depth of the queue and will only allow 8 requests to be outstanding until the TS signals an entry has been removed.

As mentioned previously, child threads have higher priority over root threads. Once TS selects a child thread to dispatch, it follows these steps:

1. TS forwards the interface descriptor pointer to the L1 interface descriptor cache (a small fully associated cache containing up to 4 interface descriptors). The interface descriptor is either found in the cache or a corresponding request is forwarded to the L2 cache. Interface descriptors return back to TS in requesting order.
2. TS then builds the transparent header but not the R0 header.
3. Finally, TS makes a thread request to the Thread Dispatcher.
4. Once the dispatch is done, TS can forget the child – unlike roots, no bookkeeping is done that has to be updated when the child retires.

If more data needs to be transferred between a parent thread and its child thread than that can fit in a single URB payload, extra data must be communicated via shared memory through data port.

1.5.1.8 Arbitration between Root and Child Threads

When both root thread queue and child thread queue are both non-empty, TS serves the child thread queue. In other words, child threads have higher priority over root threads. The only condition that the child thread queue is stalled by the root thread queue is that the head of child thread queue is a root-synchronization message and the head of root thread queue is not a synchronized root thread.

1.5.1.9 Persistent Root Thread

Persistent Root Thread (PRT) is in the context of multi-context scheduling, where the thread supports midstream interruptability for fine grain context switch. A persistent root thread in general stays in the system for a long period of time. It is normally a parent thread, and only one PRT is allowed in the system at a time. Upon context switch interrupt, instead of proceeding to completion, a PRT can save its software context and terminate. The PRT can be restarted later, *even if it had completed normally the last time it was executed*. Therefore, the PRT must always save enough context (via data port messages to a predefined surface) to allow it to restart from where it left off (including determining that it has nothing left to do). However, since only one PRT can execute at a time, once the next PRT starts, the previous one will never be restarted, thus the context save surface can be reused from one PRT to the next.

A PRT may check the Thread Restart Enable bit in the R0 header to find out whether it is a fresh start or resumed from a previous interrupt and then can continue operations from that previously saved context.

PRT can be interleaved with other root (such as parent root thread, or synchronized root thread) and child threads. A parent root thread is not necessarily a PRT, and doesn't have to be as long as it can be finished in deterministic time that is shorter than required for fine-grain context switch interrupt.

Use of PRT must follow the following rule:

- There can only be one PRT in the media pipeline at a given time. That means, there shall not be any other media primitive commands (MEDIA_OBJECT or MEDIA_OBJECT_EX) between it and the previous MI_FLUSH command. In other words, when multiple such PRTs are used in a sequence of media primitive commands, MI_FLUSH must be inserted.

1.5.2 Interfaces

1.5.2.1 Interface to VFE

TS receives an interface descriptor pointer and a URB handle from VFE. It uses the interface descriptor pointer to fetch the interface descriptor. TS uses the information in the interface descriptor along with the URB handle to fill out the transparent header in the message to TD for all threads. For root thread, TS also generate the R0 header.

TS transmits URB handle dereference signal to VFE. As described previously, the derefernce signal may be at dispatch time or at later time depending on Children Present. No matter which case, there is one and only one URB handle dereference for a thread.

1.5.2.2 Interface to Thread Dispatcher

TS creates the transparent header, assembles the URB handles and calls TD to dispatch a new thread. For an unsynchronized root thread, there is one URB handle managed by VFE and optionally one Constant URB handle managed by CS. For a synchronized root thread, there is one URB handle managed by VFE, a URB handle created by the synchronizing thread (the one that sends the 'spawn root thread' message, and optionally one Constant URB handle managed by CS. For a child thread, there is one URB handle managed by the parent thread plus an optional Constant URB handle.

1.6 Media State Model [DevSNB+]

The media state model is based on in-line state load mechanism. VFE state, URB configuration and Interface Descriptors are loaded to VFE hardware through state commands.

All Interface Descriptors have the same size and are organized as a contiguous array in memory. They can be selected by Interface Descriptor Index for a given kernel. This allows different kinds of kernels to coexist in the system.

1.7 Media State and Primitive Commands

1.7.1 MEDIA_VFE_STATE Command [DevSNB+]

An MI_FLUSH is required before MEDIA_VFE_STATE unless the only bits that are changed are scoreboard related: Scoreboard Enable, Scoreboard Type, Scoreboard Mask, Scoreboard * Delta. For these scoreboard related states, a MEDIA_STATE_FLUSH is sufficient.

- MEDIA_STATE_FLUSH (optional, only if barrier dependency is needed)
- MEDIA_INTERFACE_DESCRIPTOR_LOAD (optional)

DWord	Bit	Description
0	31:29	**Command Type =** GFXPIPE = 3h
	28:16	**Media Command Opcode =** MEDIA_VFE_STATE Pipeline[28:27] = Media = 2h; Opcode[26:24] = 0h; Subopcode[23:16] **=** 0h
	15:0	**DWord Length** (Excludes DWords 0,1) = 06h
1	31:10	**Scratch Space Base Pointer.** Specifies the 1k-byte aligned address offset to scratch space for use by the kernel. This pointer is relative to the **General State Base Address**. Format = GeneralStateOffset[31:10]
	9:8	Reserved: MBZ
	7:4	Stack Size Range = [0,11] indicating [1KBytes, 2MBytes]
	3:0	**Per Thread Scratch Space.** Specifies the amount of scratch space allowed to be used by each thread. The driver must allocate enough contiguous scratch space, pointed to by the Scratch Space Pointer, to ensure that the Maximum Number of Threads each get Per Thread Scratch Space size without exceeding the driver-allocated scratch space. **Note:** The definition of this field is different from that in 3D fixed functions, where the per-thread scratch space is specified in powers of 2. Format = U4 Range = [0,11] indicating [1k bytes, 12k bytes] [DevSNB]

DWord	Bit	Description
2	31:16	**Maximum Number of Threads.** Specifies the maximum number of simultaneous root threads allowed to be active. Used to avoid using up the scratch space, or to avoid potential deadlock. Note that MSB will be zero due to the range limit below. Format = U16 representing (thread count – 1) Range = [0, n-1] where n = (# EUs) * (# threads/EU). See *Graphics Processing Engine* for listing of #EUs and #threads in each device.
	15:8	**Number of URB Entries.** Specifies the number of URB entries that are used by the unit. Format = U8 Range = [0,64] → [0,64] Entries [DevSNB]
	7	**Reset Gateway Timer.** This field controls the reset of the timestamp counter maintained in Message Gateway. 0 – Maintaining the existing timestamp state 1 – Resetting relative timer and latching the global timestamp
	6	**Bypass Gateway Control.** This field configures Gateway to use a simple message protocol. 0 – Maintaining OpenGateway/ForwardMsg/CloseGateway protocol (legacy mode) 1 – Bypassing OpenGateway/CloseGateway protocol
	5	**Fast Preempt.** This field controls when an preempt signal is sent (broadcast) to EUs upon a context switch interrupt. If this field is set to 1, TS hardware will send the preempt signal to EUs, when the media pipe is ready (drained). If this field is set to 0, TS hardware will wait until the number of outstanding thread to be 1 (only the _PRT) before sending the preempt signal to EUs. 0 – Preempt only _PRT 1 – Fast Preempt
	4:3	**Reserved: MBZ**
	2	**Reserved:MBZ**
	1:0	**Reserved**
3	31:8	**Reserved**
	7:0	Reserved : MBZ
4	31:16	**URB Entry Allocation Size.** Specifies the length of each URB entry used by the unit, in 256-bit register increments. ROB address for URB starts after CURBE Allocated region Format = U12 Range = [0, 991] 256-bit register increments. ROB has 32KB of storage. (URB Entry Allocation Size * Number of URB Entries) + CURBE Allocation Size + 32 must be less than or equal to 1024 [DevSNB+]

DWord	Bit	Description
	15:0	**CURBE Allocation Size.** Specifies the total length allocated for CURBE, in 256-bit register increments – 1. ROB address for CURBE starts at address 32. Format = U12 Range = [0, 991] 256-bit register increments. ROB has 32KB of storage. (URB Entry Allocation Size * Number of URB Entries) + CURBE Allocation Size + 32 must be less than 1024 [Dev-SNB+] Range = [0, 2015] 256-bit register increments. ROB has 64KB of storage. (URB Entry Allocation Size * Number of URB Entries) + CURBE Allocation Size + 32 must be less than 2048 [DevSNB+]
5	31	**Scoreboard Enable.** This field enables and disables the hardware scoreboard in the Media Pipeline. If this field is cleared, hardware ignores the following scoreboard state fields. 0 – Scoreboard disabled 1 – Scoreboard enabled
	30	**Scoreboard Type.** This field selects the type of scoreboard in use. 0 – Stalling scoreboard 1 – Non-stalling scoreboard
	29:8	Reserved : MBZ
	7:0	**Scoreboard Mask.** Each bit indicates the corresponding dependency scoreboard is enabled. The scoreboard is based on the relative (X, Y) distance from the current threads' (X, Y) position. **Bit n (for n = 0…7):** Score n is enabled Format = TRUE/FALSE
6	31:28	**Scoreboard 3 Delta Y.** Relative vertical distance of the dependent instance assigned to scoreboard 3, in the form of 2's compliment. Format = S3
	27:24	**Scoreboard 3 Delta X.** Relative horizontal distance of the dependent instance assigned to scoreboard 3, in the form of 2's compliment. Format = S3
	23:16	**Scoreboard 2 Delta (X, Y)**
	15:8	**Scoreboard 1 Delta (X, Y)**
	7:0	**Scoreboard 0 Delta (X, Y)**
7	31:24	**Scoreboard 7 Delta (X, Y)**
	23:16	**Scoreboard 6 Delta (X, Y)**
	15:8	**Scoreboard 5 Delta (X, Y)**
	7:0	**Scoreboard 4 Delta (X, Y)**

1.7.2 MEDIA_CURBE_LOAD Command [DevSNB+]

MEDIA_CURBE_LOAD			
Project:	[DevSNB+]	**Length Bias:**	2
Desc			

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Media Command Opcode** Default Value: 2h MEDIA_CURBE_LOAD Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MEDIA_CURBE_LOAD Format: OpCode
	23:16	**Media Command Opcode** Default Value: 1h MEDIA_CURBE_LOAD Format: OpCode
	15:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:0	**Reserved** Project: All Format: MBZ
2	31:17	**Reserved** Project: All Format: MBZ
	16:0	**CURBE Total Data Length** Project: All Format: U17 In bytes This field provides the length in bytes of the CURBE data. This field must have the same alignment as the Curbe Object Data Start Address. It must be DQWord (32-byte) aligned. As the CURBE data are sent directly to ROB, range is limited to CURBE Allocation Size.
3	31:0	**CURBE Data Start Address** Project: All Address: DynamicBaseAddress[31:0] Surface Type: CURBE Range 0..4GB Specifies the 32-byte aligned address of the CURBE data. This pointer is relative to the **Dynamics Base Address**. It must be DWord aligned.

1.7.3 MEDIA_INTERFACE_DESCRIPTOR_LOAD Command [DevSNB+]

MEDIA_INTERFACE_DESCRIPTOR_LOAD				
Project:	[DevSNB+]		**Length Bias:**	2
Desc				
DWord	**Bit**	**Description**		
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode		
	28:27	**Media Command Opcode** Default Value: 2h MEDIA_INTERFACE_DESCRIPTOR_LOAD Format: OpCode		
	26:24	**Media Command Opcode** Default Value: 0h MEDIA_INTERFACE_DESCRIPTOR_LOAD Format: OpCode		
	23:16	**Media Command Opcode** Default Value: 2h MEDIA_INTERFACE_DESCRIPTOR_LOAD Format: OpCode		
	15:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All		
1	31:5	**Reserved** Project: All Format: MBZ		
	4:0	**Reserved** Project: All Format: MBZ		
2	31:17	**Reserved** Project: All Format: MBZ		
	16:0	**Interface Descriptor Total Length** Project: All Format: U17 In bytes Range: 32..992 [DevSNB] This field provides the length in bytes of the Interface Descriptor data. This field must have the same alignment as the Interface Descriptor Data Start Address. It must be DQWord (32-byte) aligned. As the Interface Descriptor data are sent directly to ROB, range is limited to CURBE Allocation Size. Range = [32 – 992] → [1- 31] interface descriptor entries [DevSNB]		

MEDIA_INTERFACE_DESCRIPTOR_LOAD		
3	31:0	**Interface Descriptor Data Start Address** Project: All Address: DynamicBaseAddress[31:0] Surface Type: INTERFACE_DESCRIPTOR_DATA Range: 0..4GB Specifies the 32-byte aligned address of the Interface Descriptor data. This pointer is relative to the **Dynamics Base Address**.

1.7.4 INTERFACE_DESCRIPTOR_DATA [DevSNB+]

Interface Descriptor Data payload as pointed by the Interface Descriptor Data Start Address:

INTERFACE_DESCRIPTOR_DATA		
Project: [DevSNB+] **Default Value:** 00000000h		
Desc		
DWord	**Bit**	**Description**
0	31:6	**Kernel Start Pointer** Project: All Address: InstructionBaseAddress[31:6] Surface Type: Kernel Specifies the 64-byte aligned address offset of the first instruction in the kernel. This pointer is relative to the **Instruction Base Address**.
	5:0	**Reserved** Project: All Format: MBZ
1	31:26	**Reserved** Project: All Format: MBZ
	25:20	**Reserved** Project: All Format: MBZ
	19	**Reserved** Project: All Format: MBZ
	18	**Single Program Flow (SPF)** Project: All Specifies whether the kernel program has a single program flow (SIMDnxm with m = 1) or multiple program flows (SIMDnxm with m > 1).

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

INTERFACE_DESCRIPTOR_DATA

	17	**Thread Priority** Project: All Specifies the priority of the thread for dispatch

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

	16	**Floating Point Mode** Project: All Specifies the floating point mode used by the dispatched thread.

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

	15:14	**Reserved** Project: All Format: MBZ
	13	**Illegal Opcode Exception Enable** Project: All Format: Enable This bit gets loaded into EU CR0.1[12] (note the bit # difference). See *Exceptions* and *ISA Execution Environment*.
	12	**Reserved** Project: All Format: MBZ
	11	**MaskStack Exception Enable** Project: All Format: Enable This bit gets loaded into EU CR0.1[11]. See *Exceptions* and *ISA Execution Environment*.
	10:8	**Reserved** Project: All Format: MBZ
	7	**Software Exception Enable** Project: All Format: Enable This bit gets loaded into EU CR0.1[13] (note the bit # difference). See *Exceptions* and *ISA Execution Environment*.
	6:0	**Reserved** Project: All Format: MBZ
2	31:5	**Sampler State Pointer** Project: All Address: DynamicStateOffset[31:5] Surface Type: SAMPLER_STATE Specifies the 32-byte aligned address offset of the sampler state table. This pointer is relative to the **Dynamic State Base Address**. *This field is ignored for child threads.*

INTERFACE_DESCRIPTOR_DATA

	4:2	**Sampler Count** Project: All Format: U3 Range: 0..4 Specifies how many samplers (in multiples of 4) the kernel uses. Used only for prefetching the associated sampler state entries. *This field is ignored for child threads.* *If this field is not zero, sampler state is prefetched for the first instance of a root thread upon the startup of the media pipeline.*

Value	Name	Description	Project
0h	No samplers used		All
1h	Between 1 and 4 samplers used		All
2h	Between 5 and 8 samplers used		All
3h	Between 9 and 12 samplers used		All
4h	Between 13 and 16 samplers used		All

	1:0	**Reserved** Project: All Format: MBZ
3	31:5	**Binding Table Pointer** Project: All Address: SurfaceStateOffset[31:5] – [DevSNB] Surface Type: BINDING_TABLE Specifies the 32-byte aligned address of the binding table. This pointer is relative to the **Surface State Base Address**. *This field is ignored for child threads.*
	4:0	**Binding Table Entry Count** Project: All Format: U5 Range: 0..31 Specifies how many binding table entries the kernel uses. Used only for prefetching of the binding table entries and associated surface state. **Note:** The maximum number of prefetched binding table entries is limited to 31. For kernels using a large number of binding table entries, it may be wise to set this field to zero to avoid prefetching too many entries and thrashing the state cache. *This field is ignored for child threads.* *If this field is not zero, binding table and surface state are prefetched for the first instance of a root thread upon the startup of the media pipeline.*

INTERFACE_DESCRIPTOR_DATA		
4	31:16	**Constant URB Entry Read Length** Project: All Format: U6 Range: 0..63 Specifies the amount of URB data read and passed in the thread payload for the Constant URB entry, in 8-DW register increments. A value 0 means that no Constant URB Entry will be loaded. The Constant URB Entry Read Offset field will then be ignored.
	15:0	**Constant URB Entry Read Offset** Project: All Format: U6 Range 0..2015 Specifies the offset (in 8-DW units) at which Constant URB data is to be read from the URB before being included in the thread payload. Range = [0,2015] indicating [0,2015] 256-bit register increments. ROB has 64KB of storage; 2048 entries. However, lowest 32 entries are reserved for VFE/TS to store interface descriptor data. Hence, URB Entry Read Offset plus Read Length shall not exceed 2015.
5	31:4	**Reserved** Project: [DevSNB] Format: MBZ
	3:0	**Barrier ID** Project: [DevSNB] Format: U4 Range 0..15 Specifies the barrier id that is associated with this interface descriptor
6	31:8	**Reserved** Project: All Format: MBZ
	7:0	**Reserved**
7	31:0	**Reserved** Project: All Format: MBZ

1.7.5 MEDIA_GATEWAY_STATE Command [DevSNB]

MEDIA_GATEWAY_STATE			
Project:	[DevSNB]	**Length Bias:**	2
This command updates the Message Gateway state. In particular, it updates the state for a selected Barrier.			

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

MEDIA_GATEWAY_STATE		
	28:27	**Media Command Opcode** Default Value: 2h MEDIA_GATEWAY_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MEDIA_GATEWAY_STATE Format: OpCode
	23:16	**Media Command Opcode** Default Value: 3h MEDIA_GATEWAY_STATE Format: OpCode
	15:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:24	**Reserved** Project: All Format: MBZ
	23:16	**BarrierID** Project: All Format: U4 This field indicates which one from the 16 Barrier States is updated.
	15:8	**Barrier.Byte** Project: All Format: U8 This is the initial value to be delivered by the Message Gateway to the requester threads when the Barrier Thread Count is reached.
	7:0	**Barrier.ThreadCount** Project: All Format: U8 Range 1..MaxThreadCount Barrier Thread Count is the total number of threads in a group that share the barrier function. It cannot be greater than the maximum number of threads, MaxThreadCount (DevID), supported by the product. For example, MaxThreadCount (DevSNB) = 60.

1.7.6 MEDIA_STATE_FLUSH Command [DevSNB]

MEDIA_STATE_FLUSH			
Project:	[DevSNB]	**Length Bias:**	2

This command updates the Message Gateway state. In particular, it updates the state for a selected Barrier.
This command can be considered same as a MI_Flush except that only media parser will get flushed instead of the entire 3D/media render pipeline. The command should be programmed prior to new Media state, curbe and/or interface descriptor commands when switching to a new context or programming new state for the same context.

With this command, pipelined state change is allowed for the media pipe.

It should be cautious when using this command when child_present flag in the media state is enabled. This is because that CURBE state as well as Interface Descriptor state are shared between root threads and child threads. Changing these states while child threads are generated on the fly may cause unexpected behavior.

Combining with MI_ARB_ON/OFF command, it is possible to support interruptability with the following command sequence where interrupt may be allowed only when MI_ARB_ON_OFF is ON:

```
MEDIA_STATE_FLUSH
VFE_STATE          // VFE will hold CS if watermark isn't met
MI_ARB_OFF         // There must be at least one VFE command before this one
MEDIA_OBJECT
....
MI_ARB_ON
```

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Media Command Opcode** Default Value: 2h MEDIA_STATE_FLUSH Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MEDIA_STATE_FLUSH Format: OpCode
	23:16	**Media Command Opcode** Default Value: 4h MEDIA_STATE_FLUSH Format: OpCode
	15:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All

MEDIA_STATE_FLUSH		
1	31:24	**Reserved** Project: All Format: MBZ
	23:16	**ThreadCountWaterMark** Project: All Format: U8 Zero is Valid The state flush is subject to the number of spawn threads to be dispatched to be less than or equal to the Thread Count water Mark. If the number of unoccupied threads in the system is smaller than this number, this command will stall the media pipeline. *Usage example: in certain usage a predetermined set of threads must be dispatched all together as a work group and interrupt is only allowed between groups. For that, a media state flush with the watermark set such that media pipe doesn't proceed with the next group of threads until there are enough hardware thread slots available.*
	15:0	**BarrierMask** Project: All This is a bit mask, with the bit location mapping with BarrierID, indicating whether the state flush is subject to the corresponding Barrier. When Barrier is not used, all bits must be set to 0.

Value	Name	Description	Project
0h	Not Waiting	Not waiting for the corresponding Barrier state to clear	All
1h	Waiting	Waiting for the corresponding Barrier state to clear	All

1.7.7 MEDIA_OBJECT Command [DevSNB+]

The MEDIA_OBJECT command is the basic media primitive command for the media pipeline. It supports loading of inline data as well as indirect data.

Dword	Bits	Description
0	31:29	**Command Type =** GFXPIPE = 3h
	28:16	**Media Command Opcode =** MEDIA_OBJECT Pipeline[28:27] = Media = 2h; Opcode[26:24] **=** 1h; Subopcode[23:16] = 0h
	15:0	**Dword Length** (Excludes DWords 0,1) **Generic Mode:** DWord Length = N+4, where N is in the range of [0,504]. The maximum is 504 DW (equivalent to 63 8-DW registers). When both inline and indirect data are fetched for this command, the total size in 8-DW registers must be less than 112 (with both inline data length N and indirect data length rounded up to 8-DW aligned individually). The minimal inline data length is 0.
1	31:8	**Reserved**
	7:6	**Reserved.** MBZ

Dword	Bits	Description
	5:0	**Interface Descriptor Offset.** This field specifies the offset from the interface descriptor base pointer to the interface descriptor which will be applied to this object. It is specified in units of interface descriptors. Format: U5 [DevSNB] Range = [0,30] [DevSNB only]
2	31	**Children Present.** Indicates that the root thread may send spawn messages to spawn child threads and/or synchronized root threads. If Children Present is not set, TS signals VFE to dereference the URB handle immediately after it receives acknowledgement from TD that the thread is dispatched. If Children Present is set, the URB handle is forwarded to the root thread and serves as the return URB handle for the root thread. TS does not signal deference at the time of dispatch. TS signals URB handle deference only when it receives a resource dereference message from the thread. ***In order avoid deadlock, such dereference must be issued once and only once for each URB handle.*** Format = Enable
	30:25	**Reserved.** MBZ
	24	**Thread Synchronization.** This field when set indicates that the dispatch of the thread originated from this command is based on the "spawn root thread" message. 0 = No thread synchronization 1 = Thread dispatch is synchronized by the "spawn root thread" message
	23:22	**Reserved.** MBZ
	21	**Use Scoreboard.** This field specifies whether the thread associated with this command uses hardware scoreboard. Only when this field is set, the scoreboard control fields in the VFE Dword are valid. If this field is cleared, the thread associated with this command bypasses hardware scoreboard. 0 = Not using scoreboard 1 = Using scoreboard
	20	**Reserved.** MBZ
	19	**Reserved**
	18:17	**Half-Slice Destination Select** This field selects the half slice that this thread must be sent to.

Value	Name	Description	Project
10	Half-Slice 1		
01	Half-Slice 0		
00	Either half-slice		

Dword	Bits	Description
	16:0	**Indirect Data Length.** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled – subsequently, the Indirect Data Start Address field is ignored. This field must have the same alignment as the Indirect Object Data Start Address. It must be DQWord (32-byte) aligned. As the indirect data are sent directly to URB, range is limited to 496 DW. When both inline and indirect data are fetched for this command, the total size in 8-DW registers must be less than 112 (with both inline data length and indirect data length rounded up to 8-DW aligned). Format = U17 in bytes
3	31:0	**Indirect Data Start Address.** This field specifies the Graphics Memory starting address of the data to be loaded into the kernel for processing. This pointer is relative to the **Indirect Object Base Address**. Hardware ignores this field if indirect data is not present. Alignment of this address depends on the mode of operation. It is the DWord aligned address of the indirect data. Range = [0 - 512MB] (Bits 31:29 MBZ)
4	31:25	**Reserved.** MBZ
	24:16	**Scoreboard Y** This field provides the Y term of the scoreboard value of the current thread. Format = U9
	15:9	**Reserved.** MBZ
	8:0	**Scoreboard X** This field provides the X term of the scoreboard value of the current thread. Format = U9
5	31:20	**Reserved.** MBZ
	19:16	**Scoreboard Color:** This field specifies which dependency color the current thread belongs to. It affects the dependency scoreboard control. Format = U4
	15:8	**Reserved.** MBZ
	7:0	**Scoreboard Mask:** Each bit indicates the corresponding dependency scoreboard is dependent on. This field is AND'd with the corresponding Scoreboard Mask field in the MEDIA_VFE_STATE command. **Bit n (for n = 0…7):** Scoreboard n is dependent, where bit 0 maps to n = 0. Format = TRUE/FALSE
6..N	31:0	**Inline Data** **Generic Mode:** The format of this data is specified by software. Hardware does not interpret this data; it merely passes it to the kernel for processing. The total size for the inline data and indirect data must not exceed 112 registers.

1.7.7.1 Inline Data Format in AVC-IT Mode

Each MEDIA_OBJECT_EX command in "AVC-IT mode" corresponds to the processing of one macroblock. Macroblock parameters are passed in as inline data and the non-zero DCT coefficient data (as well as motion vectors and weight/offset) for the macroblock is passed in as indirect data.

Error! Reference source not found. depicts the inline data format in AVC-IT mode. All fields in inline data are forwarded to the thread as thread payload, except the QP fields, where the derived macroblock information is filled in. Starting at GRF location, inline data are stored in GRF contiguously with the tail-end partial GRF, if present, zero-filled. Some fields are merely forwarded. Some fields are also used by VFE as indicated in the following table by a mark of [Used by VFE]. As shown, inline data starts at dword 4 of MEDIA_OBJECT_EX command.

Table 1-3 shows dwords 4+3, 4+4 and 4+5 of the inline data. Luma intra prediction mode (LumaIntraPredModes) is provided as a fixed-size data structure. Details can be found in Section 1.7.7.1.1.

Table 1-3. Inline data subfields for an Intra Macroblock in AVC-IT mode (and AVC-MC mode)

Dword	Bit	Description
4+3	31:8	Reserved
	7:0	**MbIntraStruct (Macroblock Intra Structure)** Bits: MotionVerticalFieldSelect Index 7:6: **ChromaIntraPredMode** 5: [**DevILK**] **IntraPredAvailFlagF – F** (Pixel [-1, 7] available for intra prediction) F = Is_Left_MB_Field & Is_Left_Bottom_MB_Intra 4: **IntraPredAvailFlagE – E** (left neighbor bottom half) 3: **IntraPredAvailFlagD – D** (Upper right neighbor) 2: **IntraPredAvailFlagC – C** (Upper left neighbor) 1: **IntraPredAvailFlagB – B** (Upper neighbor) 0: **IntraPredAvailFlagA – A** (Left neighbor top half)
4+4	31:16	**LumaIndraPredModes[1] (Luma Intra Prediction Modes)**
	15:0	**LumaIndraPredModes[0]**
4+5	31:16	**LumaIndraPredModes[3]**
	15:0	**LumaIndraPredModes[2]**

Dwords 4+3, 4+4 and 4+5 of the inline data for an inter-predicted macroblock is detailed in Table 1-4.

Table 1-4. Inline data subfields for an Inter Macroblock in AVC-IT mode (and AVC-MC mode)

DWord	Bit	Description
4+3	31:24	**Log2WeightDenomChroma**

DWord	Bit	Description
	23:16	**Log2WeightDenomLuma**
	15:8	**SubMbPredMode (Sub Macroblock Prediction Mode)** This field describes the prediction mode of the sub macroblocks. It contains four subfields each with 2-bits, corresponding to the 4 fixed size 8x8 sub macroblocks in sequential order. Details can be found in Section 1.7.7.1.3. This field is derived from sub_mb_type for a BP_8x8 macroblock. This field is derived from **MbType** for a non-BP_8x8 inter macroblock, and carries redundant information as **MbType**) Bits [1:0]: SubMbPredMode[0] Bits [3:2]: SubMbPredMode[1] Bits [5:4]: SubMbPredMode[2] Bits [7:6]: SubMbPredMode[3]
	7:0	**SubMbShape (Sub Macroblock Shape)** This field describes the subdivision of the sub macroblocks. It contains four subfields each with 2-bits, corresponding to the 4 fixed size 8x8 sub macroblocks in sequential order. Details can be found in Section 1.7.7.1.3. This field is derived from sub_mb_type for a BP_8x8 macroblock. This field is forced to 0 for a non-BP_8x8 inter macroblock, and effectively carries redundant information as **MbType**). Bits [1:0]: SubMbShape[0] Bits [3:2]: SubMbShape[1] Bits [5:4]: SubMbShape[2] Bits [7:6]: SubMbShape[3]
4+4	31:24	**BindingTableIndexForward – Block 3**
	23:16	**BindingTableIndexForward – Block 2**
	15:8	**BindingTableIndexForward – Block 1**
	7:0	**BindingTableIndexForward – Block 0**
4+5	31:24	**BindingTableIndexBackward – Block 3**
	23:16	**BindingTableIndexBackward – Block 2**
	15:8	**BindingTableIndexBackward – Block 1**
	7:0	**BindingTableIndexBackward – Block 0**

1.7.7.1.1 Luma Intra Prediction Modes

Luma Intra Prediction Modes (LumaIntraPredModes) is defined in Table 1-5. It is further categorized as Intra16x16PredMode (Table 1-6), Intra8x8PredMode (Table 1-7) and Intra4x4PredMode (Table 1-8),

operating on 16x16, 8x8 and 4x4 block sizes, respectively. Figure 1-8 illustrates the intra prediction directions geometrically for the Intra4x4 prediction. When a macroblock is subdivided, the intra prediction is performed for the subdivision in a predetermined order. For example, Figure 1-9 shows the block order for Intra4x4 prediction. And Figure 1-10 shows the block order of Block8x8 in a 16x16 region or Block4x4 in a 8x8 region.

Table 1-5. Definition of LumaIntraPredModes

LumaIntraPredModes [index]		Intra16x16PredMode	Intra8x8PredMode	Intra4x4PredMode
Index	**Bit**	**MbType = [1…24]** **Transform8x8Flag = 0**	**MbType = 0** **Transform8x8Flag = 1**	**MbType = 0** **Transform8x8Flag = 0**
0	15:12	MBZ	**Block8x8 3**	**Block4x4 3 (0_0)**
	11:8	MBZ	**Block8x8 2**	**Block4x4 2 (0_1)**
	7:4	MBZ	**Block8x8 1**	**Block4x4 1 (0_2)**
	3:0	**Block16x16**	**Block8x8 0**	**Block4x4 0 (0_3)**
1	15:12	MBZ	MBZ	**Block4x4 7 (1_0)**
	11:8	MBZ	MBZ	**Block4x4 6 (1_1)**
	7:4	MBZ	MBZ	**Block4x4 5 (1_2)**
	3:0	MBZ	MBZ	**Block4x4 4 (1_3)**
2	15:12	MBZ	MBZ	**Block4x4 11 (2_0)**
	11:8	MBZ	MBZ	**Block4x4 10 (2_1)**
	7:4	MBZ	MBZ	**Block4x4 9 (2 2)**
	3:0	MBZ	MBZ	**Block4x4 8 (2_3)**
3	15:12	MBZ	MBZ	**Block4x4 15 (3_0)**
	11:8	MBZ	MBZ	**Block4x4 14 (3_1)**
	7:4	MBZ	MBZ	**Block4x4 13 (3_2)**
	3:0	MBZ	MBZ	**Block4x4 12 (3_3)**

Table 1-6. Definition of Intra16x16PredMode

Intra16x16PredMode	Description
0	**Intra_16x16_Vertical**
1	**Intra_16x16_Horizontal**
2	**Intra_16x16_DC**
3	**Intra_16x16_Plane**
4 – 15	Reserved

Table 1-7. Definition of Intra8x8PredMode

Intra8x8PredMode	Description
0	**Intra_8x8_Vertical**
1	**Intra_8x8_Horizontal**
2	**Intra_8x8_DC**
3	**Intra_8x8_Diagonal_Down_Left**
4	**Intra_8x8_Diagonal_Down_Right**
5	**Intra_8x8_Vertical_Right**
6	**Intra_8x8_Horizontal_Down**
7	**Intra_8x8_Vertical_Left**
8	**Intra_8x8_Horizontal_Up**
9 – 15	Reserved

Table 1-8. Definition of Intra4x4PredMode

Intra4x4PredMode	Description
0	**Intra_4x4_Vertical**
1	**Intra_4x4_Horizontal**
2	**Intra_4x4_DC**

Intra4x4PredMode	Description
3	**Intra_4x4_Diagonal_Down_Left**
4	**Intra_4x4_Diagonal_Down_Right**
5	**Intra_4x4_Vertical_Right**
6	**Intra_4x4_Horizontal_Down**
7	**Intra_4x4_Vertical_Left**
8	**Intra_4x4_Horizontal_Up**
9 – 15	Reserved

Figure 1-8. Intra_4x4 prediction mode directions

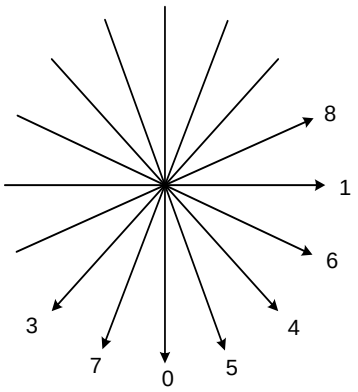


Figure 1-9. Numbers of Block4x4 in a 16x16 region

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

Figure 1-10. Numbers of Block4x4 in an 8x8 region or numbers of Block8x8 in a 16x16 region

0	1
2	3

1.7.7.1.2 Macroblock Type

Macroblock Type, MbType, is defined as a unified parameter for all slice types (I, P or B slices) as shown in Table 1-9. Furthermore, MbType has the same meaning for a P macroblock and a B macroblock. For example, BP_L0_16x16 can be viewed as a P_L0_16x16 macroblock in a P slice or a B_L0_16x16 macroblock in a B slice.

As shown in Table 1-10, Macroblock Type (MbType) is derived from *mb_type*, as defined in AVC spec, for an I-, P- or B-slice.

Table 1-9. Definition of MbType

MbType	For an Intra Macroblock (IntraMbFlag = 1)	For an Inter Macroblock (IntraMbFlag = 1)
0	I_NxN	Reserved
1	I_16x16_0_0_0	**BP**_L0_16x16
2	I_16x16_1_0_0	B_L1_16x16
3	I_16x16_2_0_0	B_Bi_16x16
4	I_16x16_3_0_0	**BP**_L0_L0_16x8
5	I_16x16_0_1_0	**BP**_L0_L0_8x16
6	I_16x16_1_1_0	B_L1_L1_16x8
7	I_16x16_2_1_0	B_L1_L1_8x16
8	I_16x16_3_1_0	B_L0_L1_16x8
9	I_16x16_0_2_0	B_L0_L1_8x16
10	I_16x16_1_2_0	B_L1_L0_16x8
11	I_16x16_2_2_0	B_L1_L0_8x16
12	I_16x16_3_2_0	B_L0_Bi_16x8
13	I_16x16_0_0_1	B_L0_Bi_8x16
14	I_16x16_1_0_1	B_L1_Bi_16x8
15	I_16x16_2_0_1	B_L1_Bi_8x16
16	I_16x16_3_0_1	B_Bi_L0_16x8
17	I_16x16_0_1_1	B_Bi_L0_8x16
18	I_16x16_1_1_1	B_Bi_L1_16x8
19	I_16x16_2_1_1	B_Bi_L1_8x16
20	I_16x16_3_1_1	B_Bi_Bi_16x8

MbType	For an Intra Macroblock (IntraMbFlag = 1)	For an Inter Macroblock (IntraMbFlag = 1)
21	I_16x16_0_2_1	B_Bi_Bi_8x16
22	I_16x16_1_2_1	**BP**_8x8
23	I_16x16_2_2_1	Reserved
24	I_16x16_3_2_1	Reserved
25	I_PCM	Reserved
26	Reserved (for SI)	Reserved
27-63	Reserved	Reserved

Table 1-10. Deriving MbType from mb_type for I, P and B slices

mb_type	I Slice		P Slice		B Slice	
	MbType	Description	MbType	Description	MbType	Description
0	0	I_NxN	1	BP_L0_16x16	22	B_Direct_16x16 mapped to BP_8x8
1	1	I_16x16_0_0_0	**4**	BP_L0_L0_16x8	1	BP_L0_16x16
2	2	I_16x16_1_0_0	**5**	BP_L0_L0_8x16	2	B_L1_16x16
3	3	I_16x16_2_0_0	**22**	BP_8x8	3	B_Bi_16x16
4	4	I_16x16_3_0_0	**22**	BP_8x8	4	BP_L0_L0_16x8
5	5	I_16x16_0_1_0	0	I_NxN	5	BP_L0_L0_8x16
6	6	I_16x16_1_1_0	1	I_16x16_0_0_0	6	B_L1_L1_16x8
7	7	I_16x16_2_1_0	2	I_16x16_1_0_0	7	B_L1_L1_8x16
8	8	I_16x16_3_1_0	3	I_16x16_2_0_0	8	B_L0_L1_16x8
9	9	I_16x16_0_2_0	4	I_16x16_3_0_0	9	B_L0_L1_8x16
10	10	I_16x16_1_2_0	5	I_16x16_0_1_0	10	B_L1_L0_16x8
11	11	I_16x16_2_2_0	6	I_16x16_1_1_0	11	B_L1_L0_8x16
12	12	I_16x16_3_2_0	7	I_16x16_2_1_0	12	B_L0_Bi_16x8

mb_type	I Slice		P Slice		B Slice	
	MbType	Description	MbType	Description	MbType	Description
13	13	I_16x16_0_0_1	8	I_16x16_3_1_0	13	B_L0_Bi_8x16
14	14	I_16x16_1_0_1	9	I_16x16_0_2_0	14	B_L1_Bi_16x8
15	15	I_16x16_2_0_1	10	I_16x16_2_0_1	15	B_L1_Bi_8x16
16	16	I_16x16_3_0_1	11	I_16x16_2_2_0	16	B_Bi_L0_16x8
17	17	I_16x16_0_1_1	12	I_16x16_3_2_0	17	B_Bi_L0_8x16
18	18	I_16x16_1_1_1	13	I_16x16_0_0_1	18	B_Bi_L1_16x8
19	19	I_16x16_2_1_1	14	I_16x16_1_0_1	19	B_Bi_L1_8x16
20	20	I_16x16_3_1_1	15	I_16x16_2_0_1	20	B_Bi_Bi_16x8
21	21	I_16x16_0_2_1	16	I_16x16_3_0_1	21	B_Bi_Bi_8x16
22	22	I_16x16_1_2_1	17	I_16x16_0_1_1	22	BP_8x8
23	23	I_16x16_2_2_1	18	I_16x16_1_1_1	0	I_NxN
24	24	I_16x16_3_2_1	19	I_16x16_2_1_1	1	I_16x16_0_0_0
25	25	I_PCM	20	I_16x16_3_1_1	2	I_16x16_1_0_0
26	n/a	n/a	21	I_16x16_0_2_1	3	I_16x16_2_0_0
27			22	I_16x16_1_2_1	4	I_16x16_3_0_0
28			23	I_16x16_2_2_1	5	I_16x16_0_1_0
29			24	I_16x16_3_2_1	6	I_16x16_1_1_0
30			25	I_PCM	7	I_16x16_2_1_0
31			n/a	n/a	8	I_16x16_3_1_0
32					9	I_16x16_0_2_0
33					10	I_16x16_2_0_1
34					11	I_16x16_2_2_0
35					12	I_16x16_3_2_0

mb_type	I Slice		P Slice		B Slice	
	MbType	Description	MbType	Description	MbType	Description
36					13	I_16x16_0_0_1
37					14	I_16x16_1_0_1
38					15	I_16x16_2_0_1
39					16	I_16x16_3_0_1
40					17	I_16x16_0_1_1
41					18	I_16x16_1_1_1
42					19	I_16x16_2_1_1
43					20	I_16x16_3_1_1
44					21	I_16x16_0_2_1
45					22	I_16x16_1_2_1
46					23	I_16x16_2_2_1
47					24	I_16x16_3_2_1
48					25	I_PCM
49-63					n/a	n/a

1.7.7.1.3 Sub Macroblock Shape and Sub Macroblock Prediction Mode

Sub Macroblock Shape, SubMbShape, describes the shape of the sub divisions of an 8x8 sub macroblock of a BP_8x8 macroblock. Sub Macroblock Prediction Mode, SubMbPredMode, indicates the prediction mode for the sub macroblock. They are defined in Table 1-11 and Table 1-12. Both of these parameters can be derived from sub_mb_type field as defined in AVC spec according to Table 1-14 and Table 1-14.

For a non-BP_8x8 inter macroblock (IntraMbFlag = 0), the sub macroblocks will be greater than and equal to 8x8. Both SubMbShape and SubMbPredMode must be filled to match with the MbType. In particular, SubMbShape is 0 and SubMbPredMode is determined based on MbType according to Table 1-15.

Table 1-11. Definition of SubMbShape for an 8x8 region of a BP_8x8 macroblock

SubMbShape	NumSubMbPart	SubMbPartWidth	SubMbPartHeight
0	1	8	8

SubMbShape	NumSubMbPart	SubMbPartWidth	SubMbPartHeight
1	2	8	4
2	2	4	8
3	4	4	4

Table 1-12. Definition of SubMbPredMode for an 8x8 region of a BP_8x8 macroblock

SubMbPredMode	Description	Comments
0	Pred_L0	P_8x8 and B_8x8
1	Pred_L1	B_8x8 only
2	BiPred	B_8x8 only
3	Reserved	

Table 1-13. Mapping sub_mb_type to SubMbType and SubMbPredMode in P macroblocks (BP_8x8)

sub_mb_type [i]	name	SubMb Prediction	SubMbPartWidth	SubMbPartHeight	SubMbShape [i]	SubMbPredMode [i]
0	P_L0_8x8	Pred_L0	8	8	**0**	**0**
1	P_L0_8x4	Pred_L0	8	4	**1**	**0**
2	P_L0_4x8	Pred_L0	4	8	**2**	**0**
3	P_L0_4x4	Pred_L0	4	4	**3**	**0**
Inferred	n/a	n/a	n/a	n/a	n/a	n/a

Table 1-14. Mapping sub_mb_type to SubMbType and SubMbPredModd in B macroblocks (BP_8x8)

sub_mb_type [i]	name	SubMb Prediction	SubMbPartWidth	SubMbPartHeight	SubMbShape [i]	SubMbPredMode [i]
0	B_Direct_8x8	Direct	4	4	**3**	
1	B_L0_8x8	Pred_L0	8	8	**0**	**0**

sub_mb_type [i]	name	SubMb Prediction	SubMbPartWidth	SubMbPartHeight	SubMbShape [i]	SubMbPredMode [i]
2	B_L1_8x8	Pred_L1	8	8	**0**	**1**
3	B_Bi_8x8	BiPred	8	8	**0**	**2**
4	B_L0_8x4	Pred_L0	8	4	**1**	**0**
5	B_L0_4x8	Pred_L0	4	8	**2**	**0**
6	B_L1_8x4	Pred_L1	8	4	**1**	**1**
7	B_L1_4x8	Pred_L1	4	8	**2**	**1**
8	B_Bi_8x4	BiPred	8	4	**1**	**2**
9	B_Bi_4x8	BiPred	4	8	**2**	**2**
10	B_L0_4x4	Pred_L0	4	4	**3**	**0**
11	B_L1_4x4	Pred_L1	4	4	**3**	**1**
12	B_Bi_4x4	BiPred	4	4	**3**	**2**
inferred	mb_type	Direct	4	4	**3**	

Table 1-15. SubMbPredMode[] for non BP_8x8 macroblocks (when IntraMbFlag = 0)

		SubMbPredMode[i]			
MbType	**Name**	**i = 0**	**i = 1**	**i = 2**	**i = 3**
0	Reserved	Reserved	Reserved	Reserved	Reserved
1	**BP**_L0_16x16	0	0	0	0
2	B_L1_16x16	1	1	1	1
3	B_Bi_16x16	2	2	2	2
4	**BP**_L0_L0_16x8	0	0	0	0
5	**BP**_L0_L0_8x16	0	0	0	0
6	B_L1_L1_16x8	1	1	1	1

		SubMbPredMode[i]			
MbType	**Name**	**i = 0**	**i = 1**	**i = 2**	**i = 3**
7	B_L1_L1_8x16	1	1	1	1
8	B_L0_L1_16x8	0	0	1	1
9	B_L0_L1_8x16	0	1	0	1
10	B_L1_L0_16x8	1	1	0	0
11	B_L1_L0_8x16	1	0	1	0
12	B_L0_Bi_16x8	0	0	2	2
13	B_L0_Bi_8x16	0	2	0	2
14	B_L1_Bi_16x8	1	1	2	2
15	B_L1_Bi_8x16	1	2	1	2
16	B_Bi_L0_16x8	2	2	0	0
17	B_Bi_L0_8x16	2	0	2	0
18	B_Bi_L1_16x8	2	2	1	1
19	B_Bi_L1_8x16	2	1	2	1
20	B_Bi_Bi_16x8	2	2	2	2
21	B_Bi_Bi_8x16	2	2	2	2

1.7.7.1.4 Motion Vector Size

In AVC, a macroblock may have 0 or 32 motion vectors and many other combinations in between. In order to simplify the AVC-IT interface, the motion vectors of a macroblock are regrouped. As shown in Table 1-16, only 5 distinct combined motion vector states (cMvState) B0, B1, B2, P3 and B3, are derived, corresponding the MvSize of 0, 2, 8, 16, and 32, respectively.

The maximum value of MvSize depends on the profile and level of the input AVC data. According to AVC Spec Table A-4 in section A.3.3.2, for Main and High Profiles at Level greater than 3.0, MinLumaBiPreSize is set to 8x8 (i.e. sub_mb_type in B macroblocks shall not be equal to B_Bi_8x4, B_Bi_4x8, or B_Bi_4x4). Therefore, B3 state is not valid for the given profile and level.

Programming Notes: *Programmers may (and should) take advantage of such profile and level restriction to conserve memory foot print for indirect data, memory bandwidth for delivering data as well as possibly the GRF register space storing motion vectors. For example, when the maximum possible MvSize is 16, only 16 dwords need to be allocated for motion vectors in both indirect data buffer and GRF space.*

Table 1-16. Motion vector regroup

Mblk Type	MV State	Max # MVs	Reference Lists	Combined MV State (cMvState)	Comments
P	P0	0	n/a	B0	Merged with B0
P	P1	1	L0	B1	Merged with B1
P	P2	4	L0	B2	Merged with B2
P	P3	**16**	L0	**P3**	Sub-macroblock partition smaller than 8x8
B	B0	**0**	n/a	**B0**	
B	B1	**2**	L0, L1, or Bi	**B1**	
B	B2	**8**	L0, L1, or Bi	**B2**	Sub-macroblock partition down to 8x8
B	B3	**32**	L0, L1, or Bi	**B3**	For a High Profile AVC data, only encountered with level <= 3.1

cMvState can be derived based on the following macroblock parameters: MbType, SubMbShape, and SubMbPredMode. Table 1-17 provides the detailed mapping.

Table 1-17. Regrouped motion vector states for an Inter Macroblock

MbType	Inter Macroblock Type	Max (sub_mb_type[])	Max (SubMBPredMode[])	Extact MV #	MV State	MvSize	Commends
1	BP_L0_16x16	n/a	n/a	1	B1	2MV	
2	B_L1_16x16	n/a	n/a	1	B1	2MV	
3	B_Bi_16x16	n/a	n/a	2	B1	2MV	
4	BP_L0_L0_16x8	n/a	n/a	2	B2	8MV	
5	BP_L0_L0_8x16	n/a	n/a	2	B2	8MV	
6	B_L1_L1_16x8	n/a	n/a	2	B2	8MV	

MbType	Inter Macroblock Type	Max (sub_mb_type[])	Max (SubMBPredMode[])	Extact MV #	MV State	MvSize	Commends
7	B_L1_L1_8x16	n/a	n/a	2	B2	8MV	
8	B_L0_L1_16x8	n/a	n/a	2	B2	8MV	
9	B_L0_L1_8x16	n/a	n/a	2	B2	8MV	
10	B_L1_L0_16x8	n/a	n/a	2	B2	8MV	
11	B_L1_L0_8x16	n/a	n/a	2	B2	8MV	
12	B_L0_Bi_16x8	n/a	n/a	3	B2	8MV	
13	B_L0_Bi_8x16	n/a	n/a	3	B2	8MV	
14	B_L1_Bi_16x8	n/a	n/a	3	B2	8MV	
15	B_L1_Bi_8x16	n/a	n/a	3	B2	8MV	
16	B_Bi_L0_16x8	n/a	n/a	3	B2	8MV	
17	B_Bi_L0_8x16	n/a	n/a	3	B2	8MV	
18	B_Bi_L1_16x8	n/a	n/a	3	B2	8MV	
19	B_Bi_L1_8x16	n/a	n/a	3	B2	8MV	
20	B_Bi_Bi_16x8	n/a	n/a	4	B2	8MV	
21	B_Bi_Bi_8x16	n/a	n/a	4	B2	8MV	
22	**BP**_8x8	0	1	4	B2	8MV	Without sub-partition, no BiPred
22	**BP**_8x8	0	2	5,6,7,8	B2	8MV	Without sub-partition, with BiPred
22	**BP**_8x8	> 0	1	5-16	P3	16MV	With sub-partition, no BiPred
22	**BP**_8x8	> 0	2	6-32	B3	32MV	With sub-partition, with BiPred

1.7.7.1.5 Binding Table Index Data in AVC-IT Mode

There are always 8 binding table indices transferred in the inline data for an Inter Macroblock, a forward and backward index for each 8x8 block in the macroblock. This data is derived from the reference index sent with each motion vector; since between 0 and 32 motion vectors can be sent, a mapping scheme is specified here to indicate which reference index is to be used for which block in the inline data.

The general scheme is that whenever the motion vectors are for partitions smaller than 8x8 then pick the upper right, since all binding table indices are guarenteed to be the same for all sub-blocks in an 8x8. If the motion vectors are for partitions larger than 8x8, then replicate the single binding table index for all 8x8s in the partition. If there is only a forward or backward motion vector specified, then replicate the binding table indices for the missing direction.

MbType	Inter Macroblock	Binding Table Replication Rule
1	**BP**_L0_16x16	L0 binding table index replicated to all 4 forward and all 4 backward
2	B_L1_16x16	L1 binding table index replicated to all 4 forward and all 4 backward
3	B_Bi_16x16	L0 replicated to all 4 forward, L1 replicated to all 4 backward
4	**BP**_L0_L0_16x8	First L0 (top) replicated to blocks 0 & 1, both forward and backward, 2nd L0 replicated to blocks 2 & 3, both forward and backward.
5	**BP**_L0_L0_8x16	First L0 (left) replicated to blocks 0 & 2, both forward and backward, 2nd L0 replicated to blocks 1 & 3, both forward and backward.
6	B_L1_L1_16x8	First L1 (top) replicated to blocks 0 & 1, both forward and backward, 2nd L1 replicated to blocks 2 & 3, both forward and backward.
7	B_L1_L1_8x16	First L1 (left) replicated to blocks 0 & 2, both forward and backward, 2nd L1 replicated to blocks 1 & 3, both forward and backward.
8	B_L0_L1_16x8	First L0 (top) replicated to blocks 0 & 1, both forward and backward, 2nd L1 replicated to blocks 2 & 3, both forward and backward.
9	B_L0_L1_8x16	First L0 (left) replicated to blocks 0 & 2, both forward and backward, 2nd L1 replicated to blocks 1 & 3, both forward and backward.
10	B_L1_L0_16x8	First L1 (top) replicated to blocks 0 & 1, both forward and backward, 2nd L0 replicated to blocks 2 & 3, both forward and backward.
11	B_L1_L0_8x16	First L1 (left) replicated to blocks 0 & 2, both forward and backward, 2nd L0 replicated to blocks 1 & 3, both forward and backward.
12	B_L0_Bi_16x8	First L0 (top) replicated to blocks 0 & 1, both forward and backward, 2nd L0 replicated to blocks forward 2 & 3, 2nd L1 to backward blocks 2 & 3
13	B_L0_Bi_8x16	First L0 (left) replicated to blocks 0 & 2, both forward and backward, 2nd L0 replicated to blocks forward 1 & 3, 2nd L1 to backward blocks 2 & 3
14	B_L1_Bi_16x8	First L1 (top) replicated to blocks 0 & 1, both forward and backward, 2nd L0 replicated to blocks forward 2 & 3, 2nd L1 to backward blocks 2 & 3
15	B_L1_Bi_8x16	First L1 (left) replicated to blocks 0 & 2, both forward and backward, 2nd L0 replicated to blocks forward 1 & 3, 2nd L1 to backward blocks 2 & 3
16	B_Bi_L0_16x8	First L0 replicated to blocks forward 0 & 1, 1st L1 to backward blocks 0 & 1, 2nd L0 replicated to blocks 2 & 3, both forward and backward
17	B_Bi_L0_8x16	First L0 replicated to blocks forward 0 & 2, 1st L1 to backward blocks 0 & 2, 2nd L0 replicated to blocks 1 & 3, both forward and backward

MbType	Inter Macroblock	Binding Table Replication Rule
18	B_Bi_L1_16x8	First L0 replicated to blocks forward 0 & 1, 1st L1 to backward blocks 0 & 1, 2nd L1 replicated to blocks 2 & 3, both forward and backward
19	B_Bi_L1_8x16	First L0 replicated to blocks forward 0 & 2, 1st L1 to backward blocks 0 & 2, 2nd L1 replicated to blocks 1 & 3, both forward and backward
20	B_Bi_Bi_16x8	First L0 replicated to forward blocks 0 & 1, 1st L1 to backward blocks 0 & 1, 2nd L0 replicated to forward blocks 2 & 3, 2nd L1 to backward blocks 2 & 3
21	B_Bi_Bi_8x16	First L0 replicated to forward blocks 0 & 2, 1st L1 to backward blocks 0 & 2, 2nd L0 replicated to forward blocks 2 & 3, 2nd L1 to backward blocks 2 & 3
22	**BP**_8x8	1) If the MvSize is 8, then the Binding Table Indices can be directly derived from the reference indices in the 8 motion vectors. 2) If MvSize is 16, then the macroblock is being split into 4x4 sub-blocks and biprediction is off (only 1 motion vector per 4x4). In this case, each 4x4 can either be forward or backward predicted, but the table reference for each set of 4 in an 8x8 is the same. Each of the 4 motion vectors in an 8x8 needs to be looked at – if one of them is forward predicted then the associated table reference can be used for that 8x8 block, and if one is backward predicted then that can be used for the backward reference for the 8x8. If all 4 motion vectors are forward, then the backward reference is not used and the forward table reference can be used as the default. 3) If MvSize is 32, then BiPred for the 4x4 sub-blocks. In this case between 4 and 8 motion vectors are sent per 8x8 block depending on whether the prediction is Bi or forward or backward. These motion vectors have to be searched in a simular way to the MvSize=16 case to find both the forward and backward reference or to replicate the existing reference if one of them is missing entirely.

1.7.7.2 Indirect Data Format in AVC-IT Mode

Indirect data in AVC-IT mode consist of Motion Vectors, Weight/Offset and Transform-domain Residue (Coefficient). All three data blocks have variable size. Sizes of Motion Vector block and the Weight-Offset block are determined by the MvSize value as shown in Table 1-18. Weight-Offset block, if present, is always packed behind the Motion Vector block. Coefficient data block can be either packed behind the Weight-Offset block or start at a predetermined offset, controlled by the fields in VFE_STATE_EX.

When coefficient data block is packed behind, it starts at the next 8-dword aligned offset from the indirect object data address. This 8-dword alignment doesn't leave any gap between the coefficient data block from the motion vector data block and weight-offset data block with one exception. When MvSize = 2 and weight-offset is not present, there is a 4-dword gap. Hardware ignores the value in the gap.

Table 1-18. Indirect subfield size in AVC-IT mode (and AVC-MC mode)

MvSize	MV		Weight/Offset		Examples
	Count	DW	Count	DW	
0	0	0	0	0	Intra macroblock in a picture containing P and/or B slices
2	2	4	1	4	P or B macroblocks with 16x16 sub macroblock
8	8	8	4	16	P or B macroblocks with minimal sub macroblock at 8x8

MvSize	MV		Weight/Offset		Examples
	Count	DW	Count	DW	
16	16	16	4	16	P macroblocks with minimal sub macroblock at less than 8x8
32	32	32	4	16	B macroblocks with minimal sub macroblock at less than 8x8

1.7.7.2.1 Motion Vector Block of Indirect Data in AVC-IT and AVC-MC Modes

Motion Vector block contains motion vectors in an intermediate format that is partially expanded according to the smallest subdivisions within an inter-predicted macroblock. During the expansion (done by AVC BSD engine or done by host software), a place that does not contain a motion vector is filled by replicating the most relevant motion vector according to the following motion vector replication rules. The intent of such motion vector replication is to allow a simpler kernel programming with fewer conditions to check. This would likely reduce the kernel footprint; however, it may or may not achieve better performance.

Motion Vector Replication Rules:

- Rule #1
    - #1.1: For L0 MV, for any partition or subpartition where there is at least one motion vector
        - If L0 inter prediction exists, the corresponding L0 MV is used
        - Else if L1 inter prediction exits (of the same block), set to the same as L1 MV
        - (Note that there is no ‘else’ here. If the partition or subpartition doesnot contain a motion vector, it will be filled according to the following replication rules)
    - #1.2: For L1 MV, for any partition or subpartition where there is at least one motion vector
        - If L1 inter prediction exists, the corresponding L1 MV is used
        - Else if L0 inter prediction exits (of the same block), set to the same as L0 MV
        - (Note that there is no ‘else’ here. If the partition or subpartition doesnot contain a motion vector, it will be filled according to the following replication rules)
- For a 16x16 partitioned macroblocked, MvSize = 2. The two MV fields follow Rule #1.
- For a macroblock with partition down to 8x8, MvSize = 8. The eight MV fields follow Rule #1.
    - For an 8x16 partition, each 8x16 is broken down into 2 8x8 stacking vertically. The 8x16 MVs (after rule #1) are replicated into both 8x8 blocks.
    - For an 16x8 partition, each 16x8 is broken down into 2 8x8 stacking horizontally. The 16x8 MVs (after rule #1) are replicated into both 8x8 blocks.
    - For an 8x8 partition, each 8x8 has its own MVs (after rule #1).
- For P macroblock with subpartition below 8x8, MvSize = 16,

- For an 8x8 partition, the 8x8 L0 MV is replicated into all the four 4x4 blocks.
- For an 4x8 subpartition within an 8x8 partition, each 4x8 is broken down into 2 4x4 stacking vertically. The 4x8 L0 MV is replicated into both 4x4 blocks.
- For an 8x4 subpartition within an 8x8 partition, each 8x4 is broken down into 2 4x4 stacking horizontally. The 8x4 MV is replicated into both 4x4 blocks.
- For a 4x4 subpartition within an 8x8 partition, each 4x4 has its own L0 MV.

- For B macroblock with subpartition below 8x8, MvSize = 32,
  - For an 8x8 partition, the 8x8 MVs (after rule #1) is replicated into all the four 4x4 blocks.
  - For an 4x8 subpartition within an 8x8 partition, each 4x8 is broken down into 2 4x4 stacking vertically. The 4x8 MVs (after rule #1) are replicated into both 4x4 blocks.
  - For an 8x4 subpartition within an 8x8 partition, each 8x4 is broken down into 2 4x4 stacking horizontally. The 8x4 MVs (after rule #1) are replicated into both 4x4 blocks.
  - For a 4x4 subpartition within an 8x8 partition, each 4x4 has its own MVs (after rule #1).

Table 1-19. Indirect data Motion Vector block in AVC-IT mode (and AVC-MC mode)

DWord	Bit	MvSize				
		0	2	8	16	32
0	31:16	n/a	MVVert_L0	MVVert_Y0_L0	MVVert_Y0_L0	MVVert_Y0_L0
	15:0	n/a	MVHorz_L0	MVHorz_Y0_L0	MVHorz_Y0_L0	MVHorz_Y0_L0
1	31:16	n/a	MVVert_L1	MVVert_Y0_L1	MVVert_Y1_L0	MVVert_Y0_L1
	15:0	n/a	MVHorz_L1	MVHorz_Y0_L1	MVHorz_Y1_L0	MVHorz_Y0_L1
2	31:0	n/a	Reserved: MBZ	MV_Y1_L0	MV_Y2_L0	MV_Y1_L0
3	31:0	n/a	Reserved: MBZ	MV_Y1_L1	MV_Y3_L0	MV_Y1_L1
4	31:0	n/a	n/a	MV_Y2_L0	MV_Y4_L0	MV_Y2_L0
5	31:0	n/a	n/a	MV_Y2_L1	MV_Y5_L0	MV_Y2_L1
6	31:0	n/a	n/a	MV_Y3_L0	MV_Y6_L0	MV_Y3_L0
7	31:0	n/a	n/a	MV_Y3_L1	MV_Y7_L0	MV_Y3_L1
8	31:0	n/a	n/a	n/a	MV_Y8_L0	MV_Y4_L0
9	31:0	n/a	n/a	n/a	MV_Y9_L0	MV_Y4_L1

DWord	Bit	MvSize				
		0	2	8	16	32
10	31:0	n/a	n/a	n/a	MV_Y10_L0	MV_Y5_L0
11	31:0	n/a	n/a	n/a	MV_Y11_L0	MV_Y5_L1
12	31:0	n/a	n/a	n/a	MV_Y12_L0	MV_Y6_L0
13	31:0	n/a	n/a	n/a	MV_Y13_L0	MV_Y6_L1
14	31:0	n/a	n/a	n/a	MV_Y14_L0	MV_Y7_L0
15	31:0	n/a	n/a	n/a	MV_Y15_L0	MV_Y7_L1
16	31:0	n/a	n/a	n/a	n/a	MV_Y8_L0
17	31:0	n/a	n/a	n/a	n/a	MV_Y8_L1
18	31:0	n/a	n/a	n/a	n/a	MV_Y9_L0
19	31:0	n/a	n/a	n/a	n/a	MV_Y9_L1
20	31:0	n/a	n/a	n/a	n/a	MV_Y10_L0
21	31:0	n/a	n/a	n/a	n/a	MV_Y10_L1
22	31:0	n/a	n/a	n/a	n/a	MV_Y11_L0
23	31:0	n/a	n/a	n/a	n/a	MV_Y11_L1
24	31:0	n/a	n/a	n/a	n/a	MV_Y12_L0
25	31:0	n/a	n/a	n/a	n/a	MV_Y12_L1
26	31:0	n/a	n/a	n/a	n/a	MV_Y13_L0
27	31:0	n/a	n/a	n/a	n/a	MV_Y13_L1
28	31:0	n/a	n/a	n/a	n/a	MV_Y14_L0
29	31:0	n/a	n/a	n/a	n/a	MV_Y14_L1
30	31:0	n/a	n/a	n/a	n/a	MV_Y15_L0
31	31:0	n/a	n/a	n/a	n/a	MV_Y15_L1

1.7.7.2.2 Weight-Offset Block of Indirect Data in AVC-IT and AVC_MC Modes for WeightedBiPredFlag ≠ 10

Table 1-20. Indirect data Weight-Offset block in AVC-IT mode (and AVC-MC mode)

DWord	Bit	MvSize		
		0	2	8, 16, 32
0	31:24	n/a	**Offset_Y_L1**	**Offset_Y_Block0_L1**
	23:16	n/a	**Weight_Y_L1**	**Weight_Y_Block0_L1**
	15:8	n/a	**Offset_Y_L0**	**Offset_Y_Block0_L0**
	7:0	n/a	**Weight_Y_L0**	**Weight_Y_Block0_L0**
1	31:16	n/a	**WO_Cb_L1**	**WO_Cb_Block0_L1**
	15:0	n/a	**WO_Cb_L0**	**WO_Cb_Block0_L0**
2	31:16	n/a	**WO_Cr_L1**	**WO_Cr_Block0_L1**
	15:0	n/a	**WO_Cr_L0**	**WO_Cr_Block0_L0**
3	31:4	n/a	Reserved: MBZ	Reserved: MBZ
	3:0		[**DevILK**] Weight is 128 [ChromaL1, Chroma L0, Luma L1, Luma L0]	[**DevILK**] Weight is 128 [ChromaL1, Chroma L0, Luma L1, Luma L0]
4	31:16	n/a	n/a	**WO_Y_Block1_L1**
	15:0	n/a	n/a	**WO_Y_Block1_L0**
5	31:16	n/a	n/a	**WO_Cb_Block1_L1**
	15:0	n/a	n/a	**WO_Cb_Block1_L0**
6	31:16	n/a	n/a	**WO_Cr_Block1_L1**
	15:0	n/a	n/a	**WO_Cr_Block1_L0**
7	31:4	n/a	n/a	Reserved: MBZ
	3:0			[**DevILK**] Weight is 128[ChromaL1, Chroma L0, Luma L1, Luma L0]
8	31:16	n/a	n/a	**WO_Y_Block2_L1**
	15:0	n/a	n/a	**WO_Y_Block2_L0**
9	31:16	n/a	n/a	**WO_Cb_Block2_L1**

DWord	Bit	MvSize		
		0	2	8, 16, 32
	15:0	n/a	n/a	**WO_Cb_Block2_L0**
10	31:16	n/a	n/a	**WO_Cr_Block2_L1**
	15:0	n/a	n/a	**WO_Cr_Block2_L0**
11	31:4	n/a	n/a	Reserved: MBZ
	3:0			[**DevILK**] Weight is 128[ChromaL1, Chroma L0, Luma L1, Luma L0]
12	31:16	n/a	n/a	**WO_Y_Block3_L1**
	15:0	n/a	n/a	**WO_Y_Block3_L0**
13	31:16	n/a	n/a	**WO_Cb_Block3_L1**
	15:0	n/a	n/a	**WO_Cb_Block3_L0**
14	31:16	n/a	n/a	**WO_Cr_Block3_L1**
	15:0	n/a	n/a	**WO_Cr_Block3_L0**
15	31:4	n/a	n/a	Reserved: MBZ
	3:0			[**DevILK**] Weight is 128[ChromaL1, Chroma L0, Luma L1, Luma L0]

1.7.7.2.3 Weight-Offset Block of Indirect Data in AVC-IT and AVC_MC Modes for WeightedBiPredFlag = 10

Implicit weights are used for B-slices when WeightedBiPredFlag = 10. In this mode the offsets are always zero and the weights are 9-bits. To fit this in the same memory footprint, the offsets are not sent and the 9-bit weights are sign extended into the 16-bit block used for the weight/offset pair in explicit mode.

Table 1-38. Indirect data Implicit Weight block in AVC-IT mode (and AVC-MC mode)

DWord	Bit	MvSize		
		0	2	8, 16, 32
	31:16	n/a	**Weight_Y_L1**	**Weight_Y_Block0_L1**
	15:0	n/a	**Weight_Y_L0**	**Weight_Y_Block0_L0**
1	31:16	n/a	**Weight_Cb_L1**	**Weight_Cb_Block0_L1**

DWord	Bit	MvSize		
		0	2	8, 16, 32
	15:0	n/a	**Weight_Cb_L0**	**Weight_Cb_Block0_L0**
2	31:16	n/a	**Weight_Cr_L1**	**Weight_Cr_Block0_L1**
	15:0	n/a	**Weight_Cr_L0**	**Weight_Cr_Block0_L0**
3	31:0	n/a	Reserved: MBZ	Reserved: MBZ
4	31:16	n/a	n/a	**Weight_Y_Block1_L1**
	15:0	n/a	n/a	**Weight_Y_Block1_L0**
5	31:16	n/a	n/a	**Weight_Cb_Block1_L1**
	15:0	n/a	n/a	**Weight_Cb_Block1_L0**
6	31:16	n/a	n/a	**Weight_Cr_Block1_L1**
	15:0	n/a	n/a	**Weight_Cr_Block1_L0**
7	31:0	n/a	n/a	Reserved: MBZ
8	31:16	n/a	n/a	**Weight_Y_Block2_L1**
	15:0	n/a	n/a	**Weight_Y_Block2_L0**
9	31:16	n/a	n/a	**Weight_Cb_Block2_L1**
	15:0	n/a	n/a	**Weight_Cb_Block2_L0**
10	31:16	n/a	n/a	**Weight_Cr_Block2_L1**
	15:0	n/a	n/a	**Weight_Cr_Block2_L0**
11	31:0	n/a	n/a	Reserved: MBZ
12	31:16	n/a	n/a	**Weight_Y_Block3_L1**
	15:0	n/a	n/a	**Weight_Y_Block3_L0**
13	31:16	n/a	n/a	**Weight_Cb_Block3_L1**
	15:0	n/a	n/a	**Weight_Cb_Block3_L0**
14	31:16	n/a	n/a	**Weight_Cr_Block3_L1**

DWord	Bit	MvSize		
		0	2	8, 16, 32
	15:0	n/a	n/a	**Weight_Cr_Block3_L0**
15	31:0	n/a	n/a	Reserved: MBZ

The weights for MvSize = 8,16,32 are replicated is exactly the same manner as the binding table indices. See section 1.7.3.1.5 for the description of the replication method. For MvSize=2 the replication is described in the following table:

MbType	Inter Macroblock	Implicit Weight Replication Rule
1	BP_L0_16x16	L0 weight and offset replicated to L1 entries
2	B_L1_16x16	L1 weight and offset replicated to L0 entries
3	B_Bi_16x16	No replication needed.

1.7.7.2.4 Transform Residual Block of Indirect Data in AVC-IT Mode

Transform-domain residual data block in AVC-IT mode is similar to that in IS mode. Only the non-zero coefficients are present in the data buffer and they are packed in the 8x8 block sequence of Y0, Y1, Y2, Y3, Cb4 and Cr5, as shown in **Error! Reference source not found.**. When an 8x8 block is further subdivided into 4x4 subblocks, the coefficients, if present, are organized in the subblock order. The smallest subblock division is referred to as a **transform block**. The indirect data length in MEDIA_OBJECT_EX includes all the non-zero coefficients for the macroblock. It must be doubleword aligned.

Each non-zero coefficient in the indirect data buffer is contained in a doubleword-size data structure consisting of the coefficient index, end of block (EOB) flag and the fixed-point coefficient value in 2's compliment form. As shown in Table 1-21, *index* is the row major 'raster' index of the coefficient within a transform block. A coefficient is a 16-bit value in 2's complement.

Table 1-21. Structure of a transform-domain residue unit

DWord	Bit	Description
0	31:16	**Transform-Domain Residual (coefficient) Value.** This field contains the value of the non-zero transform-domain residual data in 2's compliment.
	15:7	Reserved: MBZ

DWord	Bit	Description
	6:1	**Index.** This field specifies the raster-scan address (raw address) of the coefficient within the transform block. For a coefficient at Cartesian location (row, column) = (y, x) in a transform block of width W, Index is equal to (y * W + x). For example, coefficient at location (row, column) = (0, 0) in a 4x4 transform block has an index of 0; that at (2, 3) has an index of 2*4 + 3 = 11. The valid range of this field depends on the size of the transform block. Format = U6 Range = [0, 63]
	0	**EOB (End of Block).** This field indicates whether the transform-domain residue is the last one of the current transform block.

1.7.7.3 Inline Data Format in AVC-MC Mode

Each MEDIA_OBJECT_EX command in “AVC-MC mode” corresponds to the processing of one macroblock. Macroblock parameters are passed in as inline data and the pixel-domain residual data (as well as motion vectors and weight/offset) for the macroblock is passed in as indirect data.

Inline data format in AVC-MC mode follows the exact same format like the one in AVC-IT mode. Specifically, the common fields required by VFE are at the same locations and have the same meaning.

The following table depicts the inline data format in AVC-MC mode. Unlike AVC-IT, all fields in inline data are forwarded to the thread. Starting at GRF location, inline data are stored in GRF contiguously with the tail-end partial GRF, if present, zero-filled. Some fields are merely forwarded. Some fields are also used by VFE as indicated in the following table by a mark of [Used by VFE]. As shown, inline data starts at dword 4 of MEDIA_OBJECT_EX command.

1.7.7.4 Indirect Data Format in AVC-MC Mode

Indirect data in AVC-IT mode consist of Motion Vectors, Weight/Offset and pixel-domain residual data. All three data blocks have variable size.

Sizes of Motion Vector block and the Weight-Offset block are determined by the MvSize value. They are the same as in AVC-IT mode and are depicted in Table 1-18. Weight-Offset block, if present, is always packed behind the Motion Vector block. See Section 1.7.7.2 for more details.

Residual data block must start at a predetermined offset, controlled by the fields in VFE_STATE_EX.

1.7.7.5 Inline Data Format in VC1-IT Mode

Each MEDIA_OBJECT_EX command in “VC1-IT mode” corresponds to the processing of one macroblock. Macroblock parameters, including motion vectors, are passed in as inline data and the non-zero DCT coefficient data for the macroblock is passed in as indirect data.

Error! Reference source not found. depicts the inline data format in VC1-IT mode. All fields in inline data are forwarded to the thread as thread payload. Inline data are stored in GRF contiguously with the tail-end partial GRF, if present, zero-filled. Some fields are merely forwarded. Some fields are also used by VFE as indicated in the following table by a mark of [Used by VFE]. As shown, inline data starts at dword 4 of MEDIA_OBJECT_EX command.

Table 1-22. Inline data in VC1-IT mode

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>4+0</td><td>31:28</td><td>MvFieldSelect. A bit-wise representation indicating which field in the reference frame is used as the reference field for current field. It's only used in decoding interlaced pictures.

This field is valid for non-intra macroblock only.
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>28</td><td>Forward predict of current frame/field or TOP field of interlace frame, or block 0 in 4MV mode.</td></tr>
<tr><td>29</td><td>Backward predict of current frame/field or TOP field of interlace frame, or forward predict for block 1 in 4MV mode.</td></tr>
<tr><td>30</td><td>Forward predict of BOTTOM field of interlace frame, or block 2 in 4MV mode.</td></tr>
<tr><td>31</td><td>Backward predict of BOTTOM field of interlace frame, or forward predict for block 3 in 4MV mode.</td></tr>
</table>
Each corresponding bit has the following indication.
0 = The prediction is taken from the top reference field.
1 = The prediction is taken from the bottom reference field.</td></tr>
<tr><td></td><td>27</td><td>Reserved. MBZ</td></tr>
<tr><td></td><td>26</td><td>MvFieldSelectChroma . This field specifies the polarity of reference field for chroma blocks when their motion vector is derived in Motion4MV mode for interlaced (field) picture.

Non-intra macroblock only. This field is derived from MvFieldSelect.
0 = The prediction is taken from the top reference field.
1 = The prediction is taken from the bottom reference field.</td></tr>
<tr><td></td><td>25:24</td><td>MotionType – Motion Type

For frame picture, a macroblock may only be either 00 or 10.

For interlace picture, a macroblock may be of any motion types. It can be 01 if and only if DctType is 1.

This field is 00 if and only if IntraMacroblock is 1.

00 = Intra

01 = Field Motion.

10 = Frame Motion or no motion.

Others = Reserved.</td></tr>
<tr><td></td><td>23</td><td>Reserved. MBZ</td></tr>
<tr><td></td><td>22</td><td>MvSwitch. This field specifies whether the prediction needs to be switched from forward to backward or vice versa for single directional prediction for top and bottom fields of interlace frame B macroblocks.

0 = No directional prediction switch from top field to bottom field

1 = Switch directional prediction from top field to bottom field</td></tr>
</table>

DWord	Bit	Description
	21	**DctType.** This field specifies whether the residual data is coded as field residual or frame residual for interlaced picture. This field can be 1 only if MotionType is 00 (intra) or 01 (field motion). For progressive picture, this field must be set to '0', i.e. all macrobalcoks are frame macroblock. 0 = Frame residual type. 1 = Field residual type.
	20	**OverlapTransform.** This field indicates whether overlap smoothing filter should be performed on I-block boundaries. 0 = No overlap smoothing filter. 1 = Overlap smoothing filter performed.
	19	**Motion4MV.** This field indicates whether current macroblock a progressive P picture uses 4 motion vectors, one for each luminance block. It's only valid for progressive P-picture decoding. Otherwise, it is reserved and MBZ. For example, with MotionForward is 0, this field must also be set to 0. 0 = 1MV-mode. 1 = 4MV-mode.
	18	**MotionBackward.** This field specifies whether the backward motion vector is active for B-picture. This field must be 0 if Motion4MV is 1 (no backward motion vector in 4MV-mode). 0 = No backward motion vector. 1 = Use backward motion vector(s).
	17	**MotionForward.** This field specifies whether the forward motion vector is active for P and B pictures. 0 = No forward motion vector. 1 = Use forward motion vector(s).
	16	**IntraMacroblock.** This field specifies if the current macroblock is intra-coded. When set, Coded Block Pattern is ignored and no prediction is performed (i.e., no motion vectors are used). For field motion, this field indicates whether the top field of the macroblock is coded as intra. 0 = Non-intra macroblock. 1 = Intra macroblock.
	15:12	**LumaIntra8x8Flag – Luma Intra 8x8 Flag** This field specifies whether each of the four 8x8 luminance blocks are intra or inter coded when Motion4MV is set to 4MV-Mode. Each bit corresponds to one block. "0" indicates the block is inter coded and '1' indicates the block is intra coded. When Motion4MV is not 4MV-Mode, this field is reserved and MBZ. Bit 15: Y0 Bit 14: Y1 Bit 13: Y2 Bit 12: Y3

DWord	Bit	Description
	11:6	**CBP - Coded Block Pattern** This field specifies whether the 8x8 residue blocks in the macroblock are present or not. Each bit corresponds to one block. “0” indicates residue block isn’t present, “1” indicates residue block is present. Note: For each block in an intra-coded macroblock or an intra-coded block in a P macroblock in 4MV-Mode, the corresponding CBP must be 1. Subsequently, there must be at least one coefficient (this coefficient might be zero) in the indirect data buffer associated with the bock (i.e. residue block must be present). Bit 11: Y0 Bit 10: Y1 Bit 9: Y2 Bit 8: Y3 Bit 7: Cb4 Bit 6: Cr5
	5	**ChromaIntraFlag - Derived Chroma Intra Flag** This field specifies whether the chroma blocks should be treated as intra blocks based on motion vector derivation process in 4MV mode. 0 = Chroma blocks are not coded as intra. 1 = Chroma blocks are coded as intra
	4	**LastRowFlag – Last Row Flag** This field indicates that the current macroblock belongs to the last row of the picture. This field may be used by the kernel to manage pixel output when overlap transform is on. 0 = Not in the last row 1 = In the last row
	3:0	**Reserved.** MBZ
4+1	32:26	**Reserved.** MBZ
	25:24	**OSEdgeMaskChroma** This field contains the overscan edge mask for the Chroma blocks, the bit order of this field matches the overscan edge numbers shown in Figure 1-11. The left edge masks are used by VFE hardware and the top edge masks are used by the kernel software. Bit 24: Top edge of block Cb/Cr Bit 25: Left edge of block Cb/Cr

DWord	Bit	Description
	23:16	**OSEdgeMaskLuma** This field contains the overscan edge mask for the Luma blocks, the bit order of this field matches the overscan edge numbers shown in Figure 1-11. The left edge masks are used by VFE hardware and the top edge masks are used by the kernel software. Bit 16: Top edge of block Y0 Bit 17: Top edge of block Y1 Bit 18: Top edge of block Y2 Bit 19: Top edge of block Y3 Bit 20: Left edge of block Y0 Bit 21: Left edge of block Y1 Bit 22: Left edge of block Y2 Bit 23: Left edge of block Y3 ***Programming Note****: In order to create 8 predication bits from each edge mask bit, software may first create a 0, 1 vector by using a* ***shr*** *instruction with a step shift vector like 0, 1, 2, 3 (e.g. using immediate of type :v. Then each 0 or 1 of the LSB can be repeated by an* ***and*** *instruction to create 8 bits to the flag register. Alternatively, this can be achieved with one and instruction using a CURBE constant map of bit 0 and bit 1 mask.*
	15:8	**VertOrigin (Vertical Origin)** In unit of macroblocks relative to the current picture (frame or field).
	7:0	**HorzOrigin (Horizontal Origin)** In unit of macroblocks.
4+2	31:16	**MotionVector[0].Vert**
	15:0	**MotionVector[0].Horz**
4+3	31:0	**MotionVector[1]**
4+4	31:0	**MotionVector[2]**
4+5	31:0	**MotionVector[3]**
4+6	31:0	**MotionVectorChroma** Notes: This field is derived from MotionVector[3:0] as described in the following section.

DWord	Bit	Description
4+7	32:24	**Subblock Code for Y3** [Used by VFE] The following subblock coding definition applies to all 6 subblock coding bytes. Bits 7:6 are reserved.
	23;16	**Subblock Code for Y2** [Used by VFE]
	15:8	**Subblock Code for Y1** [Used by VFE]
	7:0	**Subblock Code for Y0** [Used by VFE]
4+8	31:16	**Reserved.** MBZ
	15:8	**Subblock Code for Cr** [Used by VFE]
	7:0	**Subblock Code for Cb** [Used by VFE]

Subblock Partitioning (Bits [1:0])		Subblock Present (0 means not present, 1 means present)			
Code	**Meaning**	**Bit 2**	**Bit 3**	**Bit 4**	**Bit 5**
00	Single 8x8 block (sb0)	Sb0	Don't care	Don't care	Don't care
01	Two 8x4 subblocks (sb0-1)	Sb0	Sb1	Don't care	Don't care
10	Two 4x8 subblocks (sb0-1)	Sb0	Sb1	Don't care	Don't care
11	Four 4x4 subblocks (sb0-3)	Sb0	Sb1	Sb2	Sb3

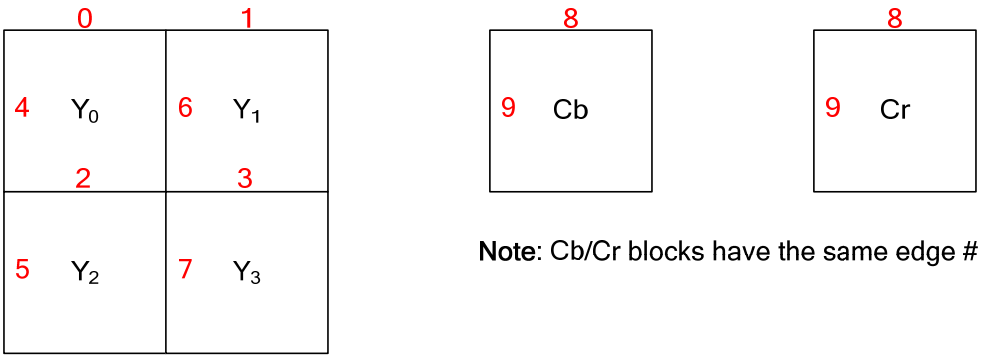


Figure 1-11 Indexing Block Edges for Overlapped Smoothing

1.7.7.5.1 Deriving Motion Vectors and Field Select for Interlaced Frame Picture

In MPEG2, the motion vectors are related to the decoded picture. For field picture, it is related to the field and which field of the reference frame that a motion vector points to is given in the bitstream by syntax

element called Motion Vector Field Select (MVFS). In contrary, motion vectors defined in VC1 standard for an interlaced frame is frame based and there is no such MVFS syntax. The VC1-IT interface defines the motion vector and MVFS following the MPEG2 convention. Therefore, motion vectors and MVFS must be derived from the frame-based motion vector values and the current macroblock position (in the top or bottom field).

The derivation of the picture-based motion vectors (luma) and MVFS is provided by the following pseudo-code. The idea is that MVFS comes from the LSB of the final pointer to the reference frame (note here it is frame based not field base). The final pointer is the addition of the frame based motion vector and the current macroblock position in the current frame (again, it is relative to the frame, not picture).

- Let (MV_X, MV_Y) be the original frame based luma motion vector in quarter-pel representation.
- Let (LMV_X, LMV_Y) be the derived field based luma motion vector and (CMV_X, CMV_Y) be the derived field based chroma motion vector, both in quarter-pel precision as well.
- Let MVFS be the derived motion vertical field select field.
- Then
    - LMV_X = MV_X;
    - if (Current_field != BOTTOM_FIELD)
        - iy = MV_Y >> 2; // Interger portion of MV_Y
    - else // Current_field == BOTTOM_FIELD
        - iy = (MV_Y >> 2) + 1; // Interger portion of MV_Y adjusted
    - MVFS = iy & 1; // 0 – top field, 1 – bottom field
    - LMV_Y = ((iy >>1)<< 2) + (MV_Y & 3);

1.7.7.5.2 Chroma Interpolations for Motion Prediction

There are two different interpolation modes are used for generating chroma samples according to the picture level parameter **bMVprecisionAndChromaRelation**: *the quarter-pel chroma motion prediction*, and *the half-pel chroma motion prediction*. The bilinear interpolation is applied for both cases.

For the quarter-pel case, the motion vectors for the chroma components are derived from the corresponding luma motion vectors according to the following pseudocodes.

For the case of 1-MV,

```
cmv.x = (mv.x + (mv.x&3==3))>>1;

cmv.y = (mv.y + (mv.y&3==3))>>1;
```

For the case of 4-MV,

```
switch(number of inter-coded blocks)

case 3: // median of three
```

```
    if(mv0.x<mv1.x && mv0.x<mv2.x)
      mv0.x = (mv1.x<mv2.x) ? mv1.x : mv2.x;
    else if(mv0.x>mv1.x && mv0.x>mv2.x)
      mv0.x = (mv1.x>mv2.x) ? mv1.x : mv2.x;
    cmv.x = (mv0.x + (mv0.x&3==3))>>1;
    cmv.y = (mv0.y + (mv0.y&3==3))>>1;
    break;
  case 4: // average of middle two
    if(mv0.x<mv1.x && mv0.x<mv2.x && mv0.x<mv3.x){
      if(mv2.x<mv3.x){ mv0.x = mv2.x;  if(mv1.x>mv3.x) mv1.x=mv3.x; }
      else          { mv0.x = mv3.x;  if(mv1.x>mv2.x) mv1.x=mv2.x; }
    }
    else if(mv0.x>mv1.x && mv0.x>mv2.x && mv0.x>mv3.x){
      if(mv2.x>mv3.x){ mv0.x = mv2.x;  if(mv1.x<mv3.x) mv1.x=mv3.x; }
      else          { mv0.x = mv3.x;  if(mv1.x<mv2.x) mv1.x=mv2.x; }
    }
    else if(mv1.x<mv2.x && mv1.x<mv3.x)
      mv1.x = (mv2.x<mv3.x) ? mv2.x : mv3.x;
    else if(mv1.x>mv2.x && mv1.x>mv3.x)
      mv1.x = (mv2.x>mv3.x) ? mv2.x : mv3.x;
  case 2: // average of two
    x = (mv0.x + mv1.x)>>1; cmv.x = (x + (x&3==3))>>1;
    y = (mv0.y + mv1.y)>>1; cmv.y = (y + (y&3==3))>>1;
    break;
  case 0: case 1: chroma should be coded as intra-blocks.
  }
```

For the half–pel case, the motion vectors for the chroma components are derived from one more extra shifting operation by rounding to the nearest full-pel if they are not currently in the half-grid.

For simple and main profile, the motion vectors are truncated so that the reference block is not totally off the picture frame:

```
// For luma:
if((mv.x>>2)<-16) mv.x = -64    +(mv.x&3);
if((mv.x>>2)> PW) mv.x = (PW<<2)+(mv.x&3);
if((mv.y>>2)<-16) mv.y = -64    +(mv.y&3);
if((mv.y>>2)> PH) mv.y = (PH<<2)+(mv.y&3);

// For chroma:
if((cmv.x>>2)<- 8) cmv.x = -32     +(cmv.x&3);
if((cmv.x>>2)>CPW) cmv.x = (CPW<<2)+(cmv.x&3);
if((cmv.y>>2)<- 8) cmv.y = -32     +(cmv.y&3);
if((cmv.y>>2)>CPH) cmv.y = (CPH<<2)+(cmv.y&3);
```

1.7.7.6 Indirect Data Format in VC1-IT Mode

Indirect data format in VC1-IT mode is identical to the transform-domain residual data block portion of the indirect data format in AVC-IT mode.

The indirect data start address in MEDIA_OBJECT_EX specifies the doubleword aligned address of the first non-zero transform-domain residue (referred to as 'coefficient') of the first block of the macroblock. The indirect data length in MEDIA_OBJECT_EX includes all the non-zero coefficients for the macroblock. It must be doubleword aligned.

Each non-zero coefficient in the indirect data buffer is contained in a doubleword-size data structure as shown in Table 1-21.

1.7.7.7 Inline Data Format in Generic Mode

MEDIA_OBJECT_EX command can also be used in "Generic mode" in place of MEDIA_OBJECT command. The only difference of the usage is to allow interface descriptor remap. MEDIA_OBJECT_EX command cannot be used together with MEDIA_OBJECT command.

1.7.8 MEDIA_OBJECT_PRT Command

The MEDIA_OBJECT_PRT command is for generating Persistent Root Thread for the media pipeline. It only supports loading of inline data but not indirect data.

This command should be used for a root thread that might have to be present in the system for a period longer than the certain minimal context-switch interrupt latency. It has to honor the context interrupt signal to terminate upon request. It should also handle replay from the interrupted point upon context restore (the same thread being dispatched more than once). In contrary, if a thread is not a Persistent Root Thread, if dispatched, it must run to completion.

The command can be used in all VFE modes, except VLD mode.

[DevSNB+]:

MEDIA_OBJECT_PRT			
Project:	[DevSNB+]	**Length Bias:**	2
For simplification, _PRT command has a fixed size of 16 DWORD			

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Media Command Opcode** Default Value: 2h MEDIA_OBJECT_PRT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h MEDIA_OBJECT_PRT Format: OpCode
	23:16	**Media Command Opcode** Default Value: 2h MEDIA_OBJECT_PRT Format: OpCode
	15:0	**DWord Length** Default Value: 14h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All **Note:** Regardless of the mode, inline data must be present in this command. The command size must fit within 16 dwords.
1	31:6	**Reserved** Project: All Format: MBZ
	5:0	**Interface Descriptor Offset** Project: All Format: U5 [DevSNB] This field specifies the offset from the interface descriptor base pointer to the interface descriptor which will be applied to this object. It is specified in units of interface descriptors.
2	31	**Children Present** Project: All Format: Enable Indicates that the root thread may send spawn messages to spawn child threads and/or synchronized root threads. If Children Present is not set, TS signals VFE to dereference the URB handle immediately after it receives acknowledgement from TD that the thread is dispatched. If Children Present is set, the URB handle is forwarded to the root thread and serves as the return URB handle for the root thread. TS does not signal deference at the time of dispatch. TS signals URB handle deference only when it receives a resource dereference message from the thread. ***In order avoid deadlock, such de-reference must be issued once and only once for each URB handle.***

MEDIA_OBJECT_PRT

	30:24	**Reserved** Project: All Format: MBZ
	23	**PRT_Fence Needed** Project: All Format: Enable This field specifies that a PRT_Fence is generated after dispatching the thread associated with this MEDIA_OBJECT_PRT. The PRT_Fence prevents additional threads following this persistent root thread until a thread spawn message is sent. The PRT_Fence is generated on first dispatch of the persistent root, as well as on re-dispatches of the persistent root after context restore.
	22	**PRT_FenceType** Project: All This field specifies the type of fence the PRT thread uses. If this field is set to 0, the fence is set at the end of the root thread queue. It will block the dispatch of the next root thread, but allowed these root threads to be populated through VFE to the root thread queue in TS. If this field is set to 1, the fence is set at the entry of VFE, similar to the fence set by the MEDIA_STATE_FLUSH command. No more command can go into the media pipe until a thread spawn message is sent (by the PRT). This field is only valid when PRT_Fence Needed is set to 1. Otherwise, it is ignored by hardware.

Value	Name	Description	Project
0h	Root thread queue	Root thread queue fence	All
1h	VFE state flush	VFE state flush fence	All

	21:17	**Reserved** Project: All Format: MBZ was **Indirect Data Length**
	16:0	**Indirect Data Length.** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled – subsequently, the Indirect Data Start Address field is ignored. This field must have the same alignment as the Indirect Object Data Start Address. It must be DQWord (32-byte) aligned. As the indirect data are sent directly to URB, range is limited to 496 DW. When both inline and indirect data are fetched for this command, the total size in 8-DW registers must be less than 112 (with both inline data length and indirect data length rounded up to 8-DW aligned). Format = U17 in bytes
3	31:0	**Indirect Data Length** Project: All Format: MBZ was **Indirect Data Start Address**
4..15	31:0	**Inline Data** Project: All Format: U32 FormatDesc

1.7.9 MEDIA_OBJECT_WALKER Command [DevSNB+]

The MEDIA_OBJECT_WALKER command uses the hardware walker in VFE for generating threads associated with a rectangular shaped object. It only supports loading of inline data (optionally) but not indirect data. Control of scoreboards (up to 8) is implicit based on the (X, Y) address of the generated thread and the scoreboard control state.

The command can be used only in Generic modes.

When **Use Scoreboard** field is set, the (X, Y) address and the Color field of the generated thread are used in the hardware scoreboard and the thread dependencies are set by states from the MEDIA_VFE_STATE command.

One or more threads may be generated by this command. This command doesn't support indirect object load. When inline data is present, it is repeated for all threads it generates. Unlike CURBE, which requires pipeline flush for change, continued change of this kind of 'global' (in the sense of shared by multiple threads from this command) data is supported when MEDIA_OBJECT_WALKER commands are issued without a pipeline flush in between.

Dword	Bits	Description
0	31:29	**Command Type =** GFXPIPE = 3h
	28:16	**Media Command Opcode =** MEDIA_OBJECT_WALKER Pipeline[28:27] = Media = 2h; Opcode[26:24] **=** 1h; Subopcode[23:16] = 03h
	15:0	**DWord Length** (Excludes DWords 0,1) Valid range: [16…N] **Note:** If this field is greater than 15, it indicates that inline data is present. If present, inline data is common for all threads generated from this command, If this field is 15, it indicates that inline data is not present. It should be noted that unlike other media object command, inline data is optional for this command.
1	31:0	**Reserved**
	7:6	**Reserved.** MBZ
	5:0	**Interface Descriptor Offset.** This field specifies the offset from the interface descriptor base pointer to the interface descriptor which will be applied to this object. It is specified in units of interface descriptors. Format: U5 [DevSNB]

Dword	Bits	Description
2	31	**Children Present.** Indicates that the root thread may send spawn messages to spawn child threads and/or synchronized root threads. If Children Present is not set, TS signals VFE to dereference the URB handle immediately after it receives acknowledgement from TD that the thread is dispatched. If Children Present is set, the URB handle is forwarded to the root thread and serves as the return URB handle for the root thread. TS does not signal deference at the time of dispatch. TS signals URB handle deference only when it receives a resource dereference message from the thread. ***In order avoid deadlock, such dereference must be issued once and only once for each URB handle.*** Format = Enable
	30:25	**Reserved. MBZ**
	24	**Thread Synchronization.** This field when set indicates that the dispatch of the thread originated from this command is based on the "spawn root thread" message. 0 = No thread synchronization 1 = Thread dispatch is synchronized by the "spawn root thread" message
	23:22	**Reserved.** MBZ
	21	**Use Scoreboard.** This field specifies whether the thread associated with this command uses hardware scoreboard. Only when this field is set, the scoreboard control fields in the VFE Dword are valid. If this field is cleared, the thread associated with this command bypasses hardware scoreboard. 0 = Not using scoreboard 1 = Using scoreboard
	20:17	**Reserved.** MBZ
	16:0	**Indirect Data Length.** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled – subsequently, the Indirect Data Start Address field is ignored. This field must have the same alignment as the Indirect Object Data Start Address. It must be DQWord (32-byte) aligned. As the indirect data are sent directly to URB, range is limited to 496 DW. When both inline and indirect data are fetched for this command, the total size in 8-DW registers must be less than or equal to 63 (with both inline data length and indirect data length rounded up to 8-DW aligned). Format = U17 in bytes
3	31:0	**Indirect Data Start Address.** This field specifies the Graphics Memory starting address of the data to be loaded into the kernel for processing. This pointer is relative to the **Indirect Object Base Address**. Hardware ignores this field if indirect data is not present. Alignment of this address depends on the mode of operation. It is the DWord aligned address of the indirect data. Range = [0 - 512MB] (Bits 31:29 MBZ)
4	31:0	**Reserved.** MBZ
5	31:8	**Reserved.** MBZ

Dword	Bits	Description
	7:0	**Scoreboard Mask:** Each bit indicates the corresponding dependency scoreboard is dependent on. This field is AND'd with the corresponding Scoreboard Mask field in the MEDIA_VFE_STATE. All threads generated by this walker command share the same dynamic mask. **Bit n (for n = 0…7):** Scoreboard n is dependent, where bit 0 maps to n = 0. Format = TRUE/FALSE
6	31	**Dual Mode** Format: TRUE / FALSE
	30	**Repel** Format: TRUE / FALSE Note: Repel should not be combined with either Dual Mode or Quad Mode
	29	**Reserved**
	28	**Reserved.** MBZ
	29:24	**Color Count Minus One.** This field specifies the number of repeat of the inner most loop of the walker. Each repeated walk position is assigned with an incremental Color number. The Color number together with the X and Y position of the thread is used for dependency scoreboard control. **Usage Example:** This allows multiple sets of dependency threads to be dispatched. Format: U4
	23:21	**Reserved.** MBZ
	20:16	**Middle Loop Extra Steps** Format = U5
	15:14	**Reserved.** MBZ
	13:12	**Local Mid-Loop Unit Y** Format = S1
	11:10	**Reserved.** MBZ
	9:8	**Mid-Loop Unit X** Format = S1
	7:0	**Reserved.** MBZ
7	31:26	**Reserved.** MBZ
	25:16	**Global Loop Exec Count** Format = U10
	15:10	**Reserved.** MBZ
	9:0	**Local Loop Exec Count** Format = U10
8	31:25	**Reserved.** MBZ

Dword	Bits	Description
	24:16	**Block Resolution Y :** Vertical resolution of the local loop. Format = U9
	15:9	**Reserved.** MBZ
	8:0	**Block Resolution X :** Horizontal resolution of the local loop. Format = U9
9	31:25	**Reserved.** MBZ
	24:16	**Local Start Y :** Starting vertical position of the local loop. Format = U9
	15:9	**Reserved.** MBZ
	8:0	**Local Start X :** Starting horizontal position of the local loop. Format = U9
10	31:25	**Reserved.** MBZ
	24:16	**[DevSNB]Local End Y :** Ending vertical position of the local loop. Format = U9
	15:9	**Reserved.** MBZ
	8:0	**[DevSNBLocal End X :** Ending horizontal position of the local loop. Format = U9
11	31:26	**Reserved.** MBZ
	25:16	**Local Outer Loop Stride Y :** Vertical stride of the local outer loop, in 2's complement. Format = S9
	15:12	**Reserved.** MBZ
	9:0	**Local Outer Loop Stride X :** Horizontal stride of the local outer loop, in 2's complement. Format = S9
112	31:26	**Reserved.** MBZ
	25:16	**Local Inner Loop Unit Y :** Vertical stride of the local inner loop, in 2's complement. Format = S9
	15:12	**Reserved.** MBZ
	9:0	**Local Inner Loop Unit X :** Horizontal stride of the local inner loop, in 2's complement. Format = S9
13	31:25	**Reserved.** MBZ
	24:16	**Global Resolution Y :** Vertical resolution of the global loop. Format = U9
	15:9	**Reserved.** MBZ

Dword	Bits	Description
	8:0	**Global Resolution X :** Horizontal resolution of the global loop. Format = U9
14	31:26	**Reserved.** MBZ
	25:16	**Global Start Y :** Starting vertical location of the global loop, in 2's complement. Format = S9
	15:10	**Reserved.** MBZ
	9:0	**Global Start X :** Starting horizontal location of the global loop, in 2's complement. Format = S9
15	31:26	**Reserved.** MBZ
	25:16	**Global Outer Loop Stride Y :** Vertical stride of the global outer loop, in 2's complement. Format = S9
	15:10	**Reserved.** MBZ
	9:0	**Global Outer Loop Stride X :** Horizontal stride of the global outer loop, in 2's complement. Format = S9
16	31:26	**Reserved.** MBZ
	25:16	**Global Inner Loop Unit Y :** Vertical stride of the global inner loop, in 2's complement. Format = S9
	15:10	**Reserved.** MBZ
	9:0	**Global Inner Loop Unit X :** Horizontal stride of the global inner loop, in 2's complement. Format = S9
17…N	31:0	**Inline Data**

1.8 Media Messages

All message formats are given in terms of dwords (32 bits) using the following conventions which are detailed in GEN4 Subsystem Chapter.

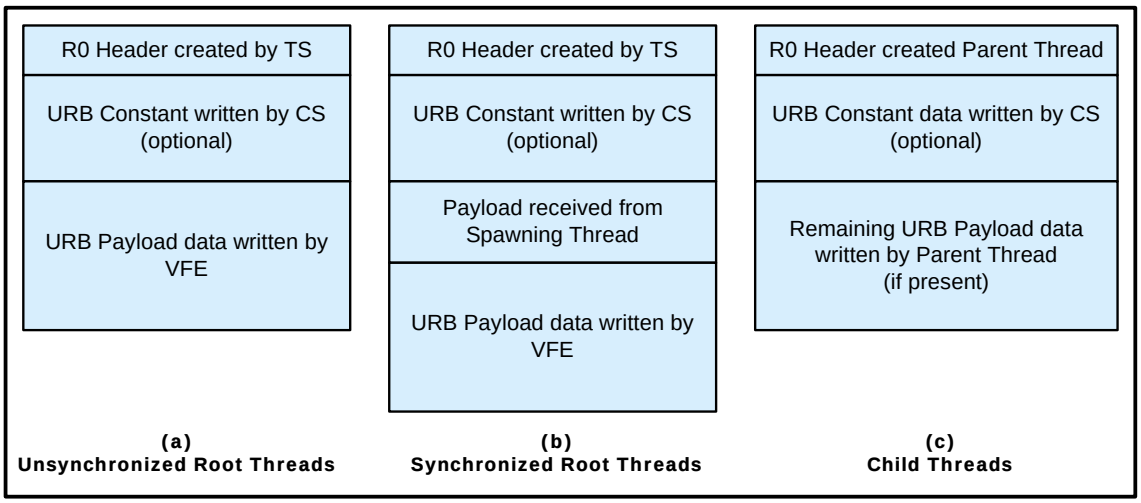
Dispatch Messages: **R**p.d

SEND Instruction Messages: **M**p.d

1.8.1 Thread Payload Messages

The root thread's register contents differ from that of child threads, as shown in Figure 1-12. The register contents for a synchronized root thread (also referred to as 'spawned root thread') and an unsynchronized one are also different. Whether the URB Constant data field is present or not is determined by the interface descriptor of a given thread. This applies to both root and child threads. When URB Constant data field is present for a synchronized root thread, URB constant data field is before the data field received from the spawning thread, which is also before the URB payload data.

Figure 1-12. Thread payload message formats for root and child threads



1.8.1.1 Generic Mode Root Thread

The following table shows the R0 register contents for a Generic mode root thread, which is generated by TS. The remaining payloads are application dependent.

Table 1-23. R0 header of a generic mode root thread

DWord	Bit	Description
R0.7	31	**Reserved**
	27:24	Reserved
	23:0	Reserved Root threads should have zero in this field.
R0.6	31:24	Reserved
	23:0	Reserved
R0.5	31:10	**Scratch Space Pointer.** Specifies the 1k-byte aligned pointer to the scratch space. This field is only valid when Scratch Space is enabled. Format = GeneralStateOffset[31:10]
	9:8	Reserved : MBZ
	7:0	**FFTID.** This ID is assigned by TS and is a unique identifier for the thread in comparison to other concurrent root threads. It is used to free up resources used by the thread upon thread completion.
R0.4	31:5	**Binding Table Pointer:** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]

DWord	Bit	Description
	4:0	Reserved : MBZ
R0.3	31:5	**Sampler State Pointer.** Specifies the 32-byte aligned pointer to the sampler state table. Format = GeneralStateOffset[31:5]
	4	Reserved : MBZ
	3:0	**Per Thread Scratch Space.** Specifies the amount of scratch space, in 16-byte quantities, allowed to be used by this thread. The value specifies the power that two will be raised to, to determine the amount of scratch space. Format = U4 Range = [0,11] indicating [1k bytes, 2M bytes] in powers of two
R0.2 [DevSNB]	31:28	Reserved : MBZ
	27:24	**BarrierID**. This field indicates which one from the 16 Barriers this kernel is associated. Format: U4
	23:16	**Barrier.Offset.** This is the offset for the Barrier to indicate the offset from the requester's RegBase (which may be 0 if Bypass Gateway Control is set to 1) for the broadcast barrier message. Barrier.Offset + RegBase must be in the valid GRF range. Otherwise, hardware behavior is undefined. It is in unit of 256-bit GRF register. The most significant bit of this field must be zero. Format = U8 Range = [0,127]
	15:9	Reserved : MBZ
	8:4	**Interface Descriptor Offset.** This field specifies the offset from the interface descriptor base pointer to the interface descriptor which will be applied to this object. It is specified in units of interface descriptors. Format = U5
	3:0	**Scoreboard Color** (only with MEDIA_OBJECT_EX)**:** This field specifies which dependency color the current thread belongs to. It affects the dependency scoreboard control. Format = U4
	3:0	**Scoreboard Color** (only with MEDIA_OBJECT_EX)**:** This field specifies which dependency color the current thread belongs to. It affects the dependency scoreboard control. Format = U4
R0.1	31:28	[DevSNB+] Reserved : MBZ
	27:26	[DevSNB+] Reserved : MBZ
	25	**Reserved.** MBZ
	24:16	[DevILK+] **Scoreboard Y** This field provides the Y term of the scoreboard value of the current thread. Format = U9

DWord	Bit	Description
	15:12	[DevSNB+] Reserved : MBZ
	11:9	**Reserved.** MBZ
	8:0	[DevILK+] **Scoreboard X** This field provides the X term of the scoreboard value of the current thread. Format = U9
R0.0	31:24	[DevSNB+] **Scoreboard Mask:** Each bit indicates the corresponding dependency scoreboard is dependent on. This field is AND'd with the corresponding Scoreboard Mask field in the MEDIA_VFE_STATE. **Bit n (for n = 0…7):** Scoreboard n is dependent, where bit 24 maps to n = 0. Format = TRUE/FALSE
	23:16	Reserved : MBZ
	15:0	**URB Handle.** This is the URB handle where indicating the URB space for use by the root thread and its children.

1.8.1.2 Root Thread from MEDIA_OBJECT_PRT [DevCTG+]

The root thread payload message for an MEDIA_OBJECT_PRT command has a fixed format independent of the VFE mode (e.g. Generic mode or AVC-IT mode). One example GRF register location is given for the condition that CURBE is disabled.

Table 1-24. Root thread payload layout for a MEDIA_OBJECT_PRT command

GRF Register	Example	Description
R0	R0	**R0 header**
R1 – R(m)	n/a	**Constants from CURBE when CURBE is enabled** m is a non-negative value
R(m+1)	R1	**In-line Data block**.

The R0 header field is as the following, which is the same as in other modes except the Thread Restart Enable bit (bit 0 of R0.2).

Table 1-25. R0 header of the thread payload of a MEDIA_OBJECT_PRT command

DWord	Bit	Description
R0.7	31	**Reserved**
	27:24	Reserved

DWord	Bit	Description
	23:0	Reserved
R0.6	31:24	**Reserved**
	23:0	**Reserved**
R0.5	31:10	**Scratch Space Pointer.** Specifies the 1k-byte aligned pointer to the scratch space. This field is only valid when Scratch Space is enabled. Format = GeneralStateOffset[31:10]
	9:8	Reserved : MBZ
	7:0	**FFTID.** This ID is assigned by TS and is a unique identifier for the thread in comparison to other concurrent root threads. It is used to free up resources used by the thread upon thread completion.
R0.4	31:5	**Binding Table Pointer:** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved : MBZ
R0.3	31:5	**Sampler State Pointer.** Specifies the 32-byte aligned pointer to the sampler state table. Format = GeneralStateOffset[31:5]
	4	Reserved : MBZ
	3:0	**Per Thread Scratch Space.** Specifies the amount of scratch space, in 16-byte quantities, allowed to be used by this thread. The value specifies the power that two will be raised to, to determine the amount of scratch space. Format = U4 Range = [0,11] indicating [1k bytes, 2M bytes] in powers of two
R0.2	31:4	**Interface Descriptor Pointer.** Specifies the 16-byte aligned pointer to *this thread's* interface descriptor. Can be used as a base from which to offset child thread's interface descriptor pointers from. Format = GeneralStateOffset[31:4]
	3:1	Reserved : MBZ
	0	**Thread Restart Enable.** If set, indicates that the persistent root thread (PRT) is being restarted, and context should be restored from the context save area before executing. Format = Enable
R0.1	31:0	Reserved : MBZ
R0.0	31:16	Reserved : MBZ
	15:0	**URB Handle.** This is the URB handle where indicating the URB space for use by the root thread and its children.

The inline data block field is the same as in the MEDIA_OBJECT_EX command with zero-filled partial GRF.

1.8.1.3 Root Thread from MEDIA_OBJECT_WALKER [DevSNB+]

The root thread payload message for an MEDIA_OBJECT_WALKER command, which must be in Generic mode, has the same format as that of the generic mode root thread format.

Table 1-26. Root thread payload layout for a MEDIA_OBJECT_WALKER command

GRF Register	Example	Description
R0	R0	**R0 header**
R1 – R(m)	n/a	**Constants from CURBE when CURBE is enabled** m is a non-negative value
R(m+1)	R1	**In-line Data block**.

The R0 header field is identical to that of Generic Mode Root Thread.

The inline data block field is the same as in the MEDIA_OBJECT command with zero-filled partial GRF.

There is no indirect data block field.

1.8.2 Thread Spawn Message

The thread spawn message is issued to the TS unit by a thread running on an EU. This message contains only one 8-DW register. The thread spawn message may be used to

- Spawn a child thread
- Spawn a root thread (start dispatching a synchronized root thread)
- Dereference URB handle
- Indicate a thread termination, dereference other TS managed resource and may or may not dereference URB handle
- Release a PRT_Fence (**[DevCTG+]**)

In order to end a root thread, the end of thread message must be targeted at the thread spawner. In this case, the root thread sends a message with a “dereference resource” in the Opcode field. The thread spawner does *not* snoop the messages sideband to determine when a root thread has ended. Thread Spawner does not track when a child thread terminates, to be consistent a child thread should also terminate with a “dereference resource” message to the Thread Spawner. Software must set the Requester Type (root or child thread) field correctly.

As TS dispatches one synchronized root thread upon receiving a ‘spawn root thread’ message (from a synchronization thread). The synchronizing thread must send the number of ‘spawn root thread’ message exactly the same as the subsequent ‘synchronized root thread’. No more, no less. Otherwise, hardware behavior is undefined.

URB Handle Offset field in this message (in M0.4) has 10 bits, allowing addressing of a large URB space. However, when a parent thread writes into the URB, it subjects to the maximum URB offset limitation of the URB write message, which is only 6 bits (see Unified Return Buffer Chapter for details). In this case, the parent thread may have to modify the URB Return Handle 0 field of the URB write message in order to subdivide the large URB space that the thread manages.

[DevIL-B+]: In addition to monitor 'end of thread message' targeted to Thread Spawner, Thread Spawner also monitors the message targeting to Message Gateway for EOT signal. Therefore, a child thread, who doesn't hold any hardware resource (URB handle or scratch memory) that Thread Spawner manages, can terminate with a Gateway message with EOT on. The reason of this new TS feature is to avoid a possible risk condition as described below.

In a system running child threads, a parent thread is monitoring the status of the child threads by communications through Message Gateway. When a child thread is about to terminate, it sends a message to the parent through Message Gateway and then sends a second message of EOT (end of thread) to TS.

There is a latency between sending a message to parent thread and the EOT to TS due to message bus arbitration. The parent thread may acknowledge the GW message and issue a new child dispatch before the EOT was processed; basically threads are issued faster than retired.

Because the messages for new child dispatch and EOT go to the same queue in TS, if the queue gets full, EOTs will get blocked. In the case when all the EUs/Threads are full, this will create a system deadlock: no EOTs can be acknowledged by TS (to free up EU resource) and no child threads can be dispatched (to free up TS queue to receive EOT message).

1.8.2.1 Message Descriptor

The following table shows the lower 16 bits of the message descriptor (lower 20 bits for **[DevIL+]**) within the SEND instruction for a thread spawn message.

Bit	Description
19	**[DevILK+]: Header Present** This bit must be set to **zero** for all Thread Spawner messages.
18:5	Reserved: MBZ
4	**Resource Select.** This field specifies the resource associated with the action taken by the Opcode. If Opcode is "Spawn thread", this field selects whether it is a child thread or a root thread. 0: spawn a *child* thread 1: spawn a *root* thread or (**[DevCTG]** only) release a PRT_Fence If Opcode == "Dereference Resource", this field indicates whether the URB handle is to be dereferenced. The URB handle can only be dereferenced once. 0: The URB handle is dereferenced 1: The URB handle is NOT dereferenced
3:2	**Reserved:** MBZ
1	**Requester Type.** This field indicates whether the requesting thread is a root thread or a child

Bit	Description
	thread. If it is a root thread, when Opcode is 0, FF managed resources will be dereferenced. If it is a child thread and Opcode is 0, no resource will be dereferenced – basically no action is required by the TS. 0: Root thread 1: Child thread
0	**Opcode.** Indicates the operation performed by the message. A root thread must terminate with a message to TS (Opcode == 0 and EOT == 1). A child thread *should* also terminate with such a message. A thread cannot terminate with an Opcode of “spawn thread”. 0: dereference resource (also used for end of thread) 1: spawn thread

1.8.2.2 Message Payload

DWord	Bit	Description
M0.7	31:0	**Reserved**:
M0.6	31:0	**Reserved**:
M0.5	31:8	Ignored
	7:0	**FFTID.** This ID is assigned by TS and is a unique identifier for the thread in comparison to other concurrent root threads. It is used to free up resources used by a root thread upon thread completion. This field is valid only if the **Opcode** is “dereference resource”, and is ignored by hardware otherwise.
M0.4	31:16	Ignored
	15:10	Dispatch URB Length. Indicates the number of 8-DW URB entries contained in the Dispatch URB Handle that will be dispatched. When spawning a child thread, the URB handle contains most of the child thread's payload including R0 header. When spawning a root thread, the URB handle contains the message passed from the requesting thread to the spawned "peer" root thread. The number of GRF registers that will be initialized at the start of the spawned child thread is the addition of this field and the number of URB constants if present. The number of GRF registers that will be initialized at the start of a spawned root thread is the addition of this field, the number of URB constants if present, and the URB handle received from VFE. This field is ignored if the Opcode is "dereference resource". Length of 0 can be used while spawning child threads to indicate that there is no payload beyond the required R0 header. Length of 0 while spawning a root thread indicates that there is no payload at all from the parent thread. A spawned root has R0 supplied by the Media_Object command indirect/inline data. Format = U6 Range = [0,63] for child threads

DWord	Bit	Description
	9:0	**URB Handle Offset.** Specifies the 8-DW URB entry offset into the URB handle that determines where the associated dispatch payload will be retrieved from when the spawned child or root thread is dispatched. This field is ignored if the **Opcode** is “dereference resource”. Format = U10 Range = [0,1023]
M0.3	31:0	Ignored
M0.2 [DevSNB+]	31:28	Ignored
	27:24	**BarrierID**. This field indicates which one from the 16 Barriers this kernel is associated. Format: U4
	23:16	Reserved
	15:10	Ignored
	9:4	**Interface Descriptor Offset.** This field specifies the offset from the interface descriptor base pointer to the interface descriptor which will be applied to this object. It is specified in units of interface descriptors. Format = U5 [DevSNB]
	3:0	**Scoreboard Color** (only with MEDIA_OBJECT_EX)**:** This field specifies which dependency color the current thread belongs to. It affects the dependency scoreboard control. Format = U4
M0.1	31:0	Ignored
M0.0	31:28	Ignored
	27:24	Reserved: MBZ
	23:16	Reserved : MBZ
	15:0	**Dispatch URB Handle** If **Opcode** (and **Requester Type**) is “spawn a child thread”: Specifies the URB handle for the child thread. If **Opcode** (and **Requester Type**) is “spawn a root thread”: Specifies the URB handle containing message (e.g. requester’s gateway information) from the requesting thread to the spawned root thread. If **Opcode** is “dereference resource”: This field is required on end of thread messages if the **Children Present** bit is set, as the handle must be dereferenced, otherwise this field is ignored.

Revision History

Revision Number	Description	Revision Date
1.0	First 2011 OpenSource edition	May 2011

§§

Intel® OpenSource HD Graphics Programmer's Reference Manual (PRM)

Volume 3 Part 1: Display Registers – VGA Registers (SandyBridge)

For the 2011 Intel Core Processor Family

May 2011

Revision 1.0

NOTICE:

This document contains information on products in the design phase of development, and Intel reserves the right to add or remove product features at any time, with or without changes to this open source documentation.

Doc Ref #: IHD-OS-V3 Pt1 – 05 11

Creative Commons License

You are free to Share — to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

No Derivative Works. You may not alter, transform, or build upon this work.

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

Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The SandyBridge chipset family, Havendale/Auburndale chipset family, Intel® 965 Express Chipset Family, Intel® G35 Express Chipset, and Intel® 965GMx Chipset Mobile Family Graphics Controller may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel® sales office or your distributor to obtain the latest specifications and before placing your product order. I2C is a two-wire communications bus/protocol developed by Philips. SMBus is a subset of the I2C bus/protocol and was developed by Intel®. Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

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

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

Copyright © 2011, Intel Corporation. All rights reserved.

Contents

1. VGA and Extended VGA Registers (00000h–00FFFh) 5

1.1 General Control and Status Registers 6
1.1.1 ST00—Input Status 0 7
1.1.2 ST01—Input Status 1 8
1.1.3 FCR—Feature Control 9
1.1.4 MSR—Miscellaneous Output 10
1.2 Sequencer Registers 11
1.2.1 SRX—Sequencer Index 12
1.2.2 SR00—Sequencer Reset 13
1.2.3 SR01—Clocking Mode 13
1.2.4 SR02—Plane/Map Mask 14
1.2.5 SR03—Character Font 15
1.2.6 SR04—Memory Mode Register 16
1.2.7 SR07—Horizontal Character Counter Reset 17
1.3 Graphics Controller Registers 17
1.3.1 GRX—GRX Graphics Controller Index Register 17
1.3.2 GR00—Set/Reset Register 18
1.3.3 GR01—Enable Set/Reset Register 18
1.3.4 GR02—Color Compare Register 19
1.3.5 GR03—Data Rotate Register 19
1.3.6 GR04—Read Plane Select Register 20
1.3.7 GR05—Graphics Mode Register 21
1.3.8 GR06—Miscellaneous Register 23
1.3.9 GR07—Color Don't Care Register 24
1.3.10 GR08—Bit Mask Register 25
1.3.11 GR10—Address Mapping 26
1.3.12 GR11—Page Selector 28
1.3.13 GR18—Software Flags 28
1.4 Attribute Controller Registers 29
1.4.1 ARX—Attribute Controller Index Register 29
1.4.2 AR[00:0F]—Palette Registers [0:F] 30
1.4.3 AR10—Mode Control Register 30
1.4.4 AR12—Memory Plane Enable Register 32
1.4.5 AR13—Horizontal Pixel Panning Register 33
1.4.6 AR14—Color Select Register 34
1.5 VGA Color Palette Registers 34
1.5.1 DACMASK—Pixel Data Mask Register 35
1.5.2 DACSTATE—DAC State Register 36
1.5.3 DACRX—Palette Read Index Register 36
1.5.4 DACWX—Palette Write Index Register 37
1.5.5 DACDATA—Palette Data Register 37
1.6 CRT Controller Register 38
1.6.1 CRX—CRT Controller Index Register 39
1.6.2 CR00—Horizontal Total Register 40
1.6.3 CR01—Horizontal Display Enable End Register 40
1.6.4 CR02—Horizontal Blanking Start Register 41
1.6.5 CR03—Horizontal Blanking End Register 41

1.6.6 CR04—Horizontal Sync Start Register 42
1.6.7 CR05—Horizontal Sync End Register 42
1.6.8 CR06—Vertical Total Register 43
1.6.9 CR07—Overflow Register (Vertical) 43
1.6.10 CR08—Preset Row Scan Register 45
1.6.11 CR09—Maximum Scan Line Register 46
1.6.12 CR0A—Text Cursor Start Register 47
1.6.13 CR0B—Text Cursor End Register 47
1.6.14 CR0C—Start Address High Register 48
1.6.15 CR0D—Start Address Low Register 48
1.6.16 CR0E—Text Cursor Location High Register 49
1.6.17 CR0F—Text Cursor Location Low Register 49
1.6.18 CR10—Vertical Sync Start Register 49
1.6.19 CR11—Vertical Sync End Register 50
1.6.20 CR12—Vertical Display Enable End Register 51
1.6.21 CR13—Offset Register 51
1.6.22 CR14—Underline Location Register 52
1.6.23 CR15—Vertical Blanking Start Register 53
1.6.24 CR16—Vertical Blanking End Register 53
1.6.25 CR17—CRT Mode Control 54
1.6.26 CR18—Line Compare Register 57
1.6.27 CR22—Memory Read Latch Data Register 57

1. VGA and Extended VGA Registers (00000h–00FFFh)

This section describes the registers and the functional operation notations for the observable registers in the VGA section. This functionality is provided as a means for support of legacy applications and operating systems. It is important to note that these registers will in general have the desired effects only when running VGA display modes.

The main exceptions to this are the palette interface which will allow real mode DOS applications and full screen VGA applications under an OS control running in high resolution (non-VGA) modes to access the palette through the VGA register mechanisms and the use of the ST01 status bits that determine when the VGA enters display enable and sync periods. Other exceptions include the register bits that control the memory accesses through the A000:0000 and B000:0000 memory segments which would get used during operating system emulation of VGA for "DOS box" applications. Some of the functions of the VGA are enabled or defeated through the programming of the VGA control register bits that which are located in the MMIO register space.

Given the legacy nature of this function, it has been adapted to the changing environment that it must operate within. The three most notable changes were the addition of high resolution display mode support, new operating system support, and the use of fixed resolution display devices (such as LCD panels). Additional control bits in the PCI Config space will affect the ability to access the registers and memory aperture associated with VGA.

Mode of Operation	VGA Disable	VGA Display	VGA Registers	Palette (VGA)	VGA Memory	VGA Banking
VGA DOS	No	Yes	Yes	Yes	Yes	No
HiRes DOS	Yes	No	Yes	Yes	No	Yes
Win9x FS DOS	Yes/No	No/Yes	Yes	Yes	Yes	Yes
Windows 9x DOS Emulation	Yes	No	Yes	Yes	Yes	Yes

VGA Display Mode	Dot Clock Select	Dot Clock Range	132 Column Text Support	9-Dot Disable Support	Main Use
Native	VGA Clock Select	25/28 MHz	No	No	Analog CRT (VGA connector)
Centered	Fixed at display Requirements	Product Specific	No	Yes	DVI connector
Upper left corner	Fixed at display Requirements	Product Specific	No	Yes	Internal Panel

Even in the native VGA display operational modes, not all combinations of bit settings result in functional operating modes. VGA display modes have the restriction that they can be used only when all other display planes are disabled except for the Cursor A when used as a popup over VGA (in devices that support that function).

In most cases, these registers can be accessed via either I/O space or memory space. The I/O space resides in the PCI compatibility hole and uses only the addresses that were part of the original VGA I/O space (which includes EGA and MDA emulation). Accesses to the VGA I/O addresses are steered to the proper bus and rely on proper setup of bridge registers. Extended VGA registers such as GR10 and GR11 use additional indexes for the already defined I/O addresses. VGA register accesses are allowed as 8 or 16 bit naturally aligned transactions only. Word transactions must have the lsb of the address set to zero. DWORD I/O operations should not be performed on these registers, **DWORD Writes are IGNORED. Reads always returns xxFFFFFFh.**

The memory space addresses listed are offsets from the base memory address programmed into the MMAPA register (PCI configuration offset 14h). For each register, the memory mapped address offset is the same address value as the I/O address. Several registers (GR10 and GR11) should never be accessed through the memory mapped aperture.

1.1 General Control and Status Registers

The setup, enable, and general registers are all directly accessible by the CPU. A sub indexing scheme is not used to read from and write to these registers.

Name	Function	Read		Write	
		I/O	Memory Offset	I/O	Memory Offset
ST00	VGA Input Status Register 0	3C2h	3C2h	—	—
ST01	VGA Input Status Register 1	3BAh/3DAh[1]	3BAh/3DAh[1]	—	—
FCR	VGA Feature Control Register	3CAh	3CAh	3BAh/3DAh[1]	3BAh/3DAh[1]
MSR	VGA Miscellaneous Output Register	3CCh	3CCh	3C2h	3C2h

Note:

1. The address selection for ST01 reads and FCR writes is dependent on CGA or MDA emulation mode as selected via the MSR register.

Various bits in these registers provide control over and the real-time status of the horizontal sync signal, the horizontal retrace interval, the vertical sync signal, and the vertical retrace interval. The horizontal retrace interval is the period during the drawing of each scan line containing active video data, when the active video data is not being displayed. This period includes the horizontal front and back porches, and the horizontal sync pulse. The horizontal retrace interval is always longer than the horizontal sync pulse. The vertical retrace interval is the period during which the scan lines not containing active video data are drawn. This includes the vertical front porch, back porch, and the vertical sync pulse. The vertical retrace interval is normally longer than the vertical sync pulse.

Display Enable is a status bit (bit 0) in VGA Input Status Register 1 that indicates when either a horizontal retrace interval or a vertical retrace interval is taking place. In the IBM* EGA graphics system (and the

ones that preceded it, including MDA and CGA), it was important to check the status of this bit to ensure that one or the other retrace intervals was taking place before reading from or writing to the frame buffer. In these earlier systems, reading from or writing to frame buffer at times outside the retrace intervals meant that the CRT controller would be denied access to the frame buffer in while accessing pixel data needed to draw pixels on the display. This resulted in either “snow” or a flickering display.

1.1.1 ST00—Input Status 0

I/O (and Memory Offset) Address: 3C2h

Default: 00h

Attributes: Read Only

7	6 5	4	3 0
CRT Interrupt	Reserved (00)	RGB Cmp / Sen	Reserved (0000)

Bit	Descriptions
7	**CRT Interrupt Pending.** This bit is here for EGA compatibility and **will always return zero**. Note that the generation of interrupts was originally enabled, through bits [4,5] of the Vertical Retrace End Register (CR11). This ability to generate interrupts at the start of the vertical retrace interval is a feature that is typically unused by DOS software and therefore is only supported through other means for use under a operating system support. 0 = CRT (vertical retrace interval) interrupt is not pending. 1 = CRT (vertical retrace interval) interrupt is pending
6:5	**Reserved.** Read as 0s.
4	**RGB Comparator / Sense.** This bit is here for compatibility and **will always return one**. Monitor detection must be done be done through the programming of registers in the MMIO space. 0 = Below threshold 1 = Above threshold
3:0	**Reserved.** Read as 0s.

1.1.2 ST01—Input Status 1

I/O (and Memory Offset) Address: 3BAh/3DAh

Default: 00h

Attributes: Read Only

The address selection is dependent on CGA or MDA emulation mode as selected via the MSR register.

7	6	5 4	3	2 1	0
Reserved (0)	Reserved (0)	Video Feedback	Vertical Retrace	Reserved (00)	Display Enable

Bit	Descriptions
7	**Reserved (as per VGA specification).** Read as 0s.
6	**Reserved.** Read as 0.
5:4	**Video Feedback 1, 0.** These are diagnostic video bits that are selected by the Color Plane Enable Register. These bits that are programmably connected to 2 of the 8 color bits sent to the palette. Bits 4 and 5 of the Color Plane Enable Register (AR12) selects which two of the 8 possible color bits become connected to these 2 bits of this register. The current software normally does not use these 2 bits. They exist for EGA compatibility.
3	**Vertical Retrace/Video.** 0 = VSYNC inactive (Indicates that a vertical retrace interval is not taking place). 1 = VSYNC active (Indicates that a vertical retrace interval is taking place). **Note:** VGA pixel generation is not locked to the display output but is loosely coupled. A VSYNC indication may not occur during the actual VSYNC going to the display but during the VSYNC that is generated as part of the VGA pixel generation. The exact relationship will vary with the VGA display operational mode. This status bit will remain active when the VGA is disabled and the device is running in high resolution modes (non-VGA) to allow for applications that (now it is incorrect) use these status registers bits. In this case, the status will come from the pipe that the VGA is assigned to. Bits 4 and 5 of the Vertical Retrace End Register (CR11) previously could program this bit to generate an interrupt at the start of the vertical retrace interval. This ability to generate interrupts at the start of the vertical retrace interval is a feature that is largely unused by legacy software. Interrupts are not supported through the VGA register bits.
2:1	**Reserved.** Read as 0s.

Bit	Descriptions
0	**Display Enable Output.** Display Enable is a status bit (bit 0) in VGA Input Status Register 1 that indicates when either a horizontal retrace interval or a vertical retrace interval is taking place. This was used with the IBM* EGA graphics system (and the ones that preceded it, including MDA and CGA). In those cases, it was important to check the status of this bit to ensure that one or the other retrace intervals was taking place before reading from or writing to the frame buffer. In these earlier systems, reading from or writing to frame buffer at times outside the retrace intervals meant that the CRT controller would be denied access to the frame buffer. This resulted in either "snow" or a flickering display. This bit provides compatibility with software designed for those early graphics controllers. This bit is currently used in DOS applications that access the palette to prevent the sparkle associated with read and write accesses to the palette ram with the same address on the same clock cycle. **This status bit will remain active when the VGA display is disabled and the device is running in high resolution modes (non-VGA) to allow for applications that (now considered incorrect) use these status registers bits. In this case, the status will come from the pipe that the VGA is assigned to. When in panel fitting VGA or centered VGA operation, the meaning of these bits will not be consistent with native VGA timings.** 0 = Active display data is being sent to the display. Neither a horizontal retrace interval or a vertical retrace interval is currently taking place. 1 = Either a horizontal retrace interval (horizontal blanking) or a vertical retrace interval (vertical blanking) is currently taking place.

1.1.3 FCR—Feature Control

I/O (and Memory Offset) Address: 3BAh/3DAh— Write; 3CAh— Read

Default: 00h

Attributes: See Address above

The I/O address used for writes is dependent on CGA or MDA emulation mode as selected via the MSR register. In the original EGA, bits 0 and 1 were used as part of the feature connector interface. Feature connector is not supported in these devices and those bits will always read as zero.

7 4	3	2 0
Reserved (0000)	VSYNC Control	Reserved (000)

Bit	Descriptions
7:4	**Reserved.** Read as 0.
3	**VSYNC Control.** This bit is provided for compatibility only and has no other function. Reads and writes to this bit have no effect other than to change the value of this bit. The previous definition of this bit selected the output on the VSYNC pin. 0 = Was used to set VSYNC output on the VSYNC pin (default). 1 = Was used to set the logical 'OR' of VSYNC and Display Enable output on the VSYNC pin. This capability was not typically very useful..
2:0	**Reserved.** Read as 0.

1.1.4 MSR—Miscellaneous Output

I/O (and Memory Offset) Address: 3C2h — Write; 3CCh— Read

Default: 00h

Attributes: See Address above

7	6	5	4	3	2	1	0
VSYNC Polarity	HSYNC Polarity	Page Select	Reserved (0)	Clock Select		Memory Enable	I/O Address

Bit	Descriptions
7	**CRT VSYNC Polarity.** This is a legacy function that is used in native VGA modes. For most cases, sync polarity will be controlled by the port control bits. The VGA settings can be optionally selected for compatibility with the original VGA when used in the VGA native mode. Sync polarity was used in VGA to signal the monitor how many lines of active display are being generated. 0 = Positive Polarity (default). 1 = Negative Polarity.
6	**CRT HSYNC Polarity.** This is a legacy function that is used in native VGA modes. For most cases, sync polarity will be controlled by the port control bits. The VGA settings can be optionally selected for compatibility with the original VGA when used in the VGA native mode. 0 = Positive Polarity (default). 1 = Negative Polarity
5	**Page Select.** In Odd/Even Memory Map Mode 1 (GR6), this bit selects the upper or lower 64 KB page in display memory for CPU access: 0 = Upper page (default) 1 = Lower page. Selects between two 64KB pages of frame buffer memory during standard VGA odd/even modes (modes 0h through 5h). Bit 1 of register GR06 can also program this bit in other modes. Note that this bit is would normally set to 1 by the software.
4	**Reserved.** Read as 0.
3:2	**Clock Select.** These bits can select the dot clock source for the CRT interface. The bits should be used to select the dot clock in standard native VGA modes only. When in the centering or upper left corner modes, these bits should be set to have no effect on the clock rate. The actual frequencies that these bits select, if they have any affect at all, is programmable through the DPLL registers that default to the standard values used for VGA. 00 = CLK0, 25.175 MHz (for standard VGA modes with 640 pixel (8-dot) horizontal resolution) (default) 01 = CLK1, 28.322 MHz. (for standard VGA modes with 720 pixel (9-dot) horizontal resolution) 10 = Was used to select an external clock (now unused) 11 = Reserved

Bit	Descriptions
1	**A0000–BFFFFh Memory Access Enable.** VGA Compatibility bit enables access to local video memory (frame buffer) at A0000–BFFFFh. When disabled, accesses to VGA memory are blocked in this region. This bit is independent of and does not block CPU access to the video linear frame buffer at other addresses. Note that it is typical for AGP chipsets to shadow this register to allow proper steering of memory accesses to the proper bus. 0 = Prevent CPU access to memory/registers/ROM through the A0000-BFFFF VGA memory aperture (default). 1 = Allow CPU access to memory/registers/ROM through the A0000-BFFFF VGA memory aperture. This memory must be mapped as UC by the CPU; see **VGA Host Access Memory Munging** in *Display and Overlay Functions*.
0	**I/O Address Select.** This bit selects 3Bxh or 3Dxh as the I/O address for the CRT Controller registers, the Feature Control Register (FCR), and Input Status Register 1 (ST01). Presently ignored (whole range is claimed), but will "ignore" 3Bx for color configuration or 3Dx for monochrome. Note that it is typical in AGP chipsets to shadow this bit and properly steer I/O cycles to the proper bus for operation where a MDA exists on another bus such as ISA. 0 = Select 3Bxh I/O address (MDA emulation) (default). 1 = Select 3Dxh I/O address (CGA emulation).

Note:

1. In standard VGA modes using the analog VGA connector, bits 7 and 6 indicate which of the three standard VGA vertical resolutions the standard VGA display should use. Extended modes, including those with a vertical resolution of 480 scan lines, may use a setting of 0 for both of these bits. Different connector standards and timing standards specify the proper use of sync polarity. This setting was "reserved" in the VGA standard.

Table 1-1. Analog CRT Display Sync Polarities

V	H	Display	Horizontal Frequency	Vertical Frequency
P	P	200 Line	15.7 KHz	60 Hz
N	P	350 Line	21.8 KHz	60 Hz
P	N	400 Line	31.5 KHz	70 Hz
N	N	480 Line	31.5 KHz	60 Hz

1.2 Sequencer Registers

The sequencer registers are accessed via either I/O space or Memory space. To access registers the VGA Sequencer Index register (SRX) at I/O address 3C4h (or memory address 3C4h) is written with the index of the desired register. Then the desired register is accessed through the data port for the sequencer registers at I/O address 3C5 (or memory address 3C5).

1.2.1 SRX—Sequencer Index

I/O (and Memory Offset) Address: 3C4h

Default: 00h

Attributes: Read/Write

7 … 3	2 … 0
Reserved (00000)	Sequencer Index

Bit	Descriptions
7:3	**Reserved.** Read as 0s.
2:0	**Sequencer Index.** This field contains a 3-bit Sequencer Index value used to access sequencer data registers at indices 0 through 7. **Notes:** • SR02 is referred to in the VGA standard as the Map Mask Register. However, the word “map” is used with multiple meanings in the VGA standard and was, therefore, deemed too confusing; hence, the reason for calling it the Plane Mask Register. • SR07 is a standard VGA register that was not documented by IBM. It is not a graphics controller extension.

1.2.2 SR00—Sequencer Reset

I/O (and Memory Offset) Address: 3C5h(Index=00h)

Default: 00h

Attributes: Read/Write

7 … 2	1	0
Reserved (000000)	Reserved	Reserved

Bit	Descriptions
7:2	**Reserved.** Read as 0.
1	**Reserved.** Reserved for VGA compatibility (was reset).
0	**Reserved.** Reserved for VGA compatibility. (was reset)

1.2.3 SR01—Clocking Mode

I/O (and Memory Offset) Address: 3C5h (Index=01h)

Default: 00h

Attributes: Read/Write

7 6	5	4	3	2	1	0
Reserved (00)	Screen Off	Shift 4	Dot Clock Divide	Shift Load	Reserved (0)	8/9 Dot Clocks

Bit	Descriptions
7:6	**Reserved.** Read as 0s.
5	**Screen Off.** 0 = Normal Operation (default). 1 = Disables video output (blanks the screen) and turns off display data fetches. Synchronization pulses to the display, however, are maintained. Setting this bit to 1 had been used as a way to more rapidly update and improve CPU access performance to the frame buffer during VGA modes. In non-VGA modes (VGA Disable=1), this bit has no effect. Before the VGA is disabled through the VGA control register, this bit should be set to stop the memory accesses from the display. **Programming Notes:** • In order to disable the VGA plane, SW must first write SR01, bit 5 = 1. It then must wait for 100us then disable the plane via Bit 31, [DevILK+] Reg. 41000). Failure to do so will cause random VGA and CPU lockups.

Bit	Descriptions
4	**Shift 4.** 0 = Load video shift registers every 1 or 2 character clocks (depending on bit 2 of this register) (default). 1 = Load shift registers every 4th character clock.
3	**Dot Clock Divide.** Setting this bit to 1 stretches doubles all horizontal timing periods that are specified in the VGA horizontal CRTC registers. This bit is used in standard VGA 40-column text modes to stretch timings to create horizontal resolutions of either 320 or 360 pixels (as opposed to 640 or 720 pixels, normally used in standard VGA 80-column text modes). The effect of this is that there will actually be twice the number of pixels sent to the display per line. 0 = Pixel clock is left unaltered (used for 640 (720) pixel modes); (default). 1 = Pixel clock divided by 2 (used for 320 (360) pixel modes).
2	**Shift Load.** Bit 4 of this register must be 0 for this bit to be effective. 0 = Load video data shift registers every character clock (default). 1 = Load video data shift registers every other character clock.
1	**Reserved.** Read as 0.
0	**8/9 Dot Clocks.** This bit determines whether a character clock is 8 or 9 dot clocks long if clock doubling is disabled and 16 or 18 clocks if it is. This also changes the interpretation of the pixel panning values (see chart). An additional control bit determines if this bit is to be ignored and 8-dot characters are to be used always. The 9-dot disable would be used when doubling the horizontal pixels on a 1280 wide display or non-doubling on a 640 wide display. Panning however will occur according to the expected outcome. 0 = 9 dot clocks (9 horizontal pixels) per character in text modes with a horizontal resolution of 720 pixels. 1 = 8 dot clocks (8 horizontal pixels) per character in text or graphics modes with a horizontal resolution of 640 pixels.

1.2.4 SR02—Plane/Map Mask

I/O (and Memory Offset) Address: 3C5h (Index=02h)

Default: 00h

Attributes: Read/Write

7 — 4	3 — 0
Reserved	Memory Planes Processor Write Access Enable.

Bit	Descriptions
7:4	**Reserved.** Read as 0s.
3:0	**Memory Planes [3:0] Processor Write Access Enable.** In both the Odd/Even Mode and the Chain 4 Mode, these bits still control access to the corresponding color plane. 0 = Disable. 1 = Enable. **Note:** This register is referred to in the VGA standard as the Map Mask Register. However, the word "map" is used with multiple meanings in the VGA standard and was, therefore, considered too confusing; hence, the reason for calling it the Plane Mask Register.

1.2.5 SR03—Character Font

I/O (and Memory Offset) Address: 3C5h (index=03h)

Default: 00h

Attributes: Read/Write

7 6	5	4	3 2	1 0
Reserved (00)	Char Map A Select (bit 0)	Char Map B Select (bit 0)	Character Map A Select (bits 2 and 1)	Character Map B Select (bits 2 and 1)

Bit	Descriptions
7:6	**Reserved.** Read as 0s.
3:2,5	**Character Map Select Bits for Character Map B.** These three bits are used to select the character map (character generator tables) to be used as the secondary character set (font). Note that the numbering of the maps is not sequential. **Bit [3:2, 5]** / **Map Number** / **Table Location** 00,0 / 0 / 1st 8KB of plane 2 at offset 0 (default) 00,1 / 4 / 2nd 8KB of plane 2 at offset 8K 01,0 / 1 / 3rd 8KB of plane 2 at offset 16K 01,1 / 5 / 4th 8KB of plane 2 at offset 24K 10,0 / 2 / 5th 8KB of plane 2 at offset 32K 10,1 / 6 / 6th 8KB of plane 2 at offset 40K 11,0 / 3 / 7th 8KB of plane 2 at offset 48K 11,1 / 7 / 8th 8KB of plane 2 at offset 56K
1:0,4	**Character Map Select Bits for Character Map A.** These three bits are used to select the character map (character generator tables) to be used as the primary character set (font). Note that the numbering of the maps is not sequential. **Bit [1:0,4]** / **Map Number** / **Table Location** 0,00 / 0 / 1st 8KB of plane 2 at offset 0 (default) 0,01 / 4 / 2nd 8KB of plane 2 at offset 8K 0,10 / 1 / 3rd 8KB of plane 2 at offset 16K 0,11 / 5 / 4th 8KB of plane 2 at offset 24K 1,00 / 2 / 5th 8KB of plane 2 at offset 32K 1,01 / 6 / 6th 8KB of plane 2 at offset 40K 1,10 / 3 / 7th 8KB of plane 2 at offset 48K 1,11 / 7 / 8th 8KB of plane 2 at offset 56K

NOTES:

1. In text modes, bit 3 of the video data's attribute byte normally controls the foreground intensity. This bit may be redefined to control switching between character sets. This latter function is

enabled whenever there is a difference in the values of the Character Font Select A and the Character Font Select B bits. If the two values are the same, the character select function is disabled and attribute bit 3 controls the foreground intensity.

2. Bit 1 of the Memory Mode Register (SR04) must be set to 1 for the character font select function of this register to be active. Otherwise, only character maps 0 and 4 are available.

1.2.6 SR04—Memory Mode Register

I/O (and Memory Offset) Address: 3C5h (index=04h)

Default: 00h

Attributes: Read/Write

7 4	3	2	1	0
Reserved (0000)	Chain 4	Odd/Even	Extended Memory	Reserved (0)

Bit	Description
7:4	**Reserved.** Read as 0.
3	**Chain 4 Mode.** The selections made by this bit affect both CPU Read and write accesses to the frame buffer. 0 = The manner in which the frame buffer memory is mapped is determined by the setting of bit 2 of this register (default). 1 = The frame buffer memory is mapped in such a way that the function of address bits 0 and 1 are altered so that they select planes 0 through 3. This setting is used in mode x13 to allow all four planes to be accessed via sequential addresses.
2	**Odd/Even Mode.** Bit 3 of this register must be set to 0 for this bit to be effective. The selections made by this bit affect only non-paged CPU accesses to the frame buffer through the VGA aperture. 0 = The frame buffer memory is mapped in such a way that the function of address bit 0 such that even addresses select planes 0 and 2 and odd addresses select planes 1 and 3 (default). 1 = Addresses sequentially access data within a bit map, and the choice of which map is accessed is made according to the value of the Plane Mask Register (SR02).
1	**Extended Memory Enable.** This bit must be set to 1 to enable the selection and use of character maps in plane 2 via the Character Map Select Register (SR03). 0 = Disable CPU accesses to more than the first 64KB of VGA standard memory (default). 1 = Enable CPU accesses to the rest of the 256KB total VGA memory beyond the first 64KB.
0	**Reserved.** Read as 0.

1.2.7 SR07—Horizontal Character Counter Reset

I/O (and Memory Offset) Address: 3C5h (index=07h)

Default: 00h

Attributes: Read/Write

For standard VGAs, writing this register (with any data) causes the horizontal character counter to be held in reset (the character counter output will remain 0). It remained in reset until a write occurred to any other sequencer register location with SRX set to an index of 0 through 6. In this implementation that sequence has no such special effect.

The vertical line counter is clocked by a signal derived from the horizontal display enable (which does not occur if the horizontal counter is held in reset). Therefore, if a write occurs to this register during the vertical retrace interval, both the horizontal and vertical counters will be set to 0. A write to any other sequencer register location (with SRX set to an index of 0 through 6) may then be used to start both counters with reasonable synchronization to an external event via software control. Although this was a standard VGA register, it was not documented by IBM.

Bit	Description
7:0	**Horizontal Character Counter.**

1.3 Graphics Controller Registers

The graphics controller registers are accessed via either I/O space or Memory space. Accesses to the registers of the VGA Graphics Controller are done through the use of address 3CEh (or memory address 3CEh) written with the index of the desired register. Then the desired register is accessed through the data port for the graphics controller registers at I/O address 3CFh (or memory address 3CFh). Indexes 10 and 11 should only be accessed through the I/O space only.

1.3.1 GRX—GRX Graphics Controller Index Register

I/O (and Memory Offset) Address: 3CEh

Default: 000UUUUUb (U=Undefined)

Attributes: Read/Write

7 5	4 0
Reserved (000)	Graphics Controller Register Index

Bit	Description
7:5	**Reserved.** Read as 0.
4:0	**Graphics Controller Register Index.** This field selects any one of the graphics controller registers (GR00-GR18]) to be accessed via the data port at I/O (or memory offset) location 3CFh.

1.3.2 GR00—Set/Reset Register

I/O (and Memory Offset) Address: 3CFh (index=00h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

7 4	3	2	1	0
Reserved (0000)	Plane 3	Plane 2	Plane 1	Plane 0

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Set/Reset Plane [3:0].** When the Write Mode bits (bits 0 and 1) of the Graphics Mode Register (GR05) are set to select Write Mode 0, all 8 bits of each byte of each memory plane are set to either 1 or 0 as specified in the corresponding bit in this register, if the corresponding bit in the Enable Set/Reset Register (GR01) is set to 1. When the Write Mode bits (bits 0 and 1) of the Graphics Mode Register (GR05) are set to select Write Mode 3, all CPU data written to the frame buffer is rotated, then logically ANDed with the contents of the Bit Mask Register (GR08), and then treated as the addressed data's bit mask, while value of these four bits of this register are treated as the color value.

1.3.3 GR01—Enable Set/Reset Register

I/O (and Memory Offset) Address: 3CFh (Index=01h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

7 4	3	2	1	0
Reserved (0000)	Enable Set/ Reset Plane 3	Enable Set/ Reset Plane 2	Enable Set/ Reset Plane 1	Enable Set/ Reset Plane 0

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Enable Set/Reset Plane [3:0].** This register works in conjunction with the Set/Reset Register (GR00). The Write Mode bits (bits 0 and 1) must be set for Write Mode 0 for this register to have any effect. 0 = The corresponding memory plane can be read from or written to by the CPU without any special bitwise operations taking place. 1 = The corresponding memory plane is set to 0 or 1 as specified in the Set/Reset Register (GR00).

1.3.4 GR02—Color Compare Register

I/O (and Memory Offset) Address: 3CFh (Index=02h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

7 – 4	3	2	1	0
Reserved (0000)	Color Compare			
	Plane 3	Plane 2	Plane 1	Plane 0

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Color Compare Plane [3:0].** When the Read Mode bit (bit 3) of the Graphics Mode Register (GR05) is set to select Read Mode 1, all 8 bits of each byte of each of the 4 memory planes of the frame buffer corresponding to the address from which a CPU read access is being performed are compared to the corresponding bits in this register (if the corresponding bit in the Color Don't Care Register (GR07) is set to 1). The value that the CPU receives from the read access is an 8-bit value that shows the result of this comparison, wherein value of 1 in a given bit position indicates that all of the corresponding bits in the bytes across all of the memory planes that were included in the comparison had the same value as their memory plane's respective bits in this register.

1.3.5 GR03—Data Rotate Register

I/O (and Memory Offset) Address: 3CFh (Index=03h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

7 – 5	4 – 3	2 – 0
Reserved	Function Select	Rotate Count

Bit	Description
7:5	**Reserved.** Read as 0.

Bit	Description
4:3	**Function Select.** These bits specify the logical function (if any) to be performed on data that is meant to be written to the frame buffer (using the contents of the memory read latch) just before it is actually stored in the frame buffer at the intended address location. 00 = Data being written to the frame buffer remains unchanged, and is simply stored in the frame buffer. 01 = Data being written to the frame buffer is logically ANDed with the data in the memory read latch before it is actually stored in the frame buffer. 10 = Data being written to the frame buffer is logically ORed with the data in the memory read latch before it is actually stored in the frame buffer. 11 = Data being written to the frame buffer is logically XORed with the data in the memory read latch before it is actually stored in the frame buffer.
2:0	**Rotate Count.** These bits specify the number of bits to the right to rotate any data that is meant to be written to the frame buffer just before it is actually stored in the frame buffer at the intended address location.

1.3.6 GR04—Read Plane Select Register

I/O (and Memory Offset) Address: 3CFh (Index=04h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

7 … 2	1 … 0
Reserved (000000)	Read Plane Select

Bit	Description
7:2	**Reserved.** Read as 0.
1:0	**Read Plane Select.** These two bits select the memory plane from which the CPU reads data in Read Mode 0. In Odd/Even Mode, bit 0 of this register is ignored. In Chain 4 Mode, both bits 1 and 0 of this register are ignored. The four memory planes are selected as follows: 00 = Plane 0 01 = Plane 1 10 = Plane 2 11 = Plane 3 These two bits also select which of the four memory read latches may be read via the Memory read Latch Data Register (CR22). The choice of memory read latch corresponds to the choice of plane specified in the table above. The Memory Read Latch Data register and this additional function served by 2 bits are features of the VGA standard that were never documented by IBM.

1.3.7 GR05—Graphics Mode Register

I/O (and Memory Offset) Address: 3CFh (Index=05h)

Default: 0UUU U0UUb (U=Undefined)

Attributes: Read/Write

7	6 5	4	3	2	1 0
Reserved (0)	Shift Register Control	Odd/Even	Read Mode	Reserved	Write Mode

Bit	Description
7	**Reserved.** Read as 0.
6:5	**Shift Register Control.** In standard VGA modes, pixel data is transferred from the 4 graphics memory planes to the palette via a set of 4 serial output bits. These 2 bits of this register control the format in which data in the 4 memory planes is serialized for these transfers to the palette. **Bits [6:5]=00** One bit of data at a time from parallel bytes in each of the 4 memory planes is transferred to the palette via the 4 serial output bits, with 1 of each of the serial output bits corresponding to a memory plane. This provides a 4-bit value on each transfer for 1 pixel, making possible a choice of 1 of 16 colors per pixel. Serial Out / 1st Xfer / 2nd Xfer / 3rd Xfer / 4th Xfer / 5th Xfer / 6th Xfer / 7th Xfer / 8th Xfer Bit 3 / plane 3 bit 7 / plane 3 bit 6 / plane 3 bit 5 / plane 3 bit 4 / plane 3 bit 3 / plane 3 bit 2 / plane 3 bit 1 / plane 3 bit 0 Bit 2 / plane 2 bit 7 / plane 2 bit 6 / plane 2 bit 5 / plane 2 bit 4 / plane 2 bit 3 / plane 2 bit 2 / plane 2 bit 1 / plane 2 bit 0 Bit 1 / plane 1 bit 7 / plane 1 bit 6 / plane 1 bit 5 / plane 1 bit 4 / plane 1 bit 3 / plane 1 bit 2 / plane 1 bit 1 / plane 1 bit 0 Bit 0 / plane 0 bit 7 / plane 0 bit 6 / plane 0 bit 5 / plane 0 bit 4 / plane 0 bit 3 / plane 0 bit 2 / plane 0 bit 1 / plane 0 bit 0 **Bits [6:5]=01** Two bits of data at a time from parallel bytes in each of the 4 memory planes are transferred to the palette in a pattern that alternates per byte between memory planes 0 and 2, and memory planes 1 and 3. First the even-numbered and odd-numbered bits of a byte in memory plane 0 are transferred via serial output bits 0 and 1, respectively, while the even-numbered and odd-numbered bits of a byte in memory plane 2 are transferred via serial output bits 2 and 3. Next, the even-numbered and odd-numbered bits of a byte in memory plane 1 are transferred via serial output bits 0 and 1, respectively, while the even-numbered and odd-numbered bits of memory plane 3 are transferred via serial out bits 1 and 3. This provides a pair of 2-bit values (one 2-bit value for each of 2 pixels) on each transfer, making possible a choice of 1 of 4 colors per pixel. Serial Out / 1st Xfer / 2nd Xfer / 3rd Xfer / 4th Xfer / 5th Xfer / 6th Xfer / 7th Xfer / 8th Xfer Bit 3 / plane 2 bit 7 / plane 2 bit 5 / plane 2 bit 3 / plane 2 bit 1 / plane 3 bit 7 / plane 3 bit 5 / plane 3 bit 3 / plane 3 bit 1 Bit 2 / plane 2 bit 6 / plane 2 bit 4 / plane 2 bit 2 / plane 2 bit 0 / plane 3 bit 6 / plane 3 bit 4 / plane 3 bit 2 / plane 3 bit 0 Bit 1 / plane 0 bit 7 / plane 0 bit 5 / plane 0 bit 3 / plane 0 bit 1 / plane 1 bit 7 / plane 1 bit 5 / plane 1 bit 3 / plane 1 bit 1 Bit 0 / plane 0 bit 6 / plane 0 bit 4 / plane 0 bit 2 / plane 0 bit 0 / plane 1 bit 6 / plane 1 bit 4 / plane 1 bit 2 / plane 1 bit 0 This alternating pattern is meant to accommodate the use of the Odd/Even mode of organizing the 4

Bit	Description
	memory planes, which is used by standard VGA modes 2h and 3h. **Bits [6:5]=1x** Four bits of data at a time from parallel bytes in each of the 4 memory planes are transferred to the palette in a pattern that iterates per byte through memory planes 0 through 3. First the 4 most significant bits of a byte in memory plane 0 are transferred via the 4 serial output bits, followed by the 4 least significant bits of the same byte. Next, the same transfers occur from the parallel byte in memory planes 1, 2 and lastly, 3. Each transfer provides either the upper or lower half of an 8 bit value for the color for each pixel, making possible a choice of 1 of 256 colors per pixel. This is the setting used in mode x13. Serial Out / 1st Xfer / 2nd Xfer / 3rd Xfer / 4th Xfer / 5th Xfer / 6th Xfer / 7th Xfer / 8th Xfer Bit 3 / plane 0 bit 7 / plane 0 bit 3 / plane 1 bit 7 / plane 1 bit 3 / plane 2 bit 7 / plane 2 bit 3 / plane 3 bit 7 / plane 3 bit 3 Bit 2 / plane 0 bit 6 / plane 0 bit 2 / plane 1 bit 6 / plane 1 bit 2 / plane 2 bit 6 / plane 2 bit 2 / plane 3 bit 6 / plane 3 bit 2 Bit 1 / plane 0 bit 5 / plane 0 bit 1 / plane 1 bit 5 / plane 1 bit 1 / plane 2 bit 5 / plane 2 bit 1 / plane 3 bit 5 / plane 3 bit 1 Bit 0 / plane 0 bit 4 / plane 0 bit 0 / plane 1 bit 4 / plane 1 bit 0 / plane 2 bit 4 / plane 2 bit 0 / plane 3 bit 4 / plane 3 bit 0 This pattern is meant to accommodate mode 13h, a standard VGA 256-color graphics mode.
4	**Odd/Even Mode.** 0 = Addresses sequentially access data within a bit map, and the choice of which map is accessed is made according to the value of the Plane Mask Register (SR02). 1 = The frame buffer is mapped in such a way that the function of address bit 0 is such that even addresses select memory planes 0 and 2 and odd addresses select memory planes 1 and 3. **Note:** This works in a way that is the inverse of (and is normally set to be the opposite of) bit 2 of the Memory Mode Register (SR02).
3	**Read Mode.** 0 = During a CPU read from the frame buffer, the value returned to the CPU is data from the memory plane selected by bits 1 and 0 of the Read Plane Select Register (GR04). 1 = During a CPU read from the frame buffer, all 8 bits of the byte in each of the 4 memory planes corresponding to the address from which a CPU read access is being performed are compared to the corresponding bits in this register (if the corresponding bit in the Color Don't Care Register (GR07) is set to 1). The value that the CPU receives from the read access is an 8-bit value that shows the result of this comparison. A value of 1 in a given bit position indicates that all of the corresponding bits in the bytes across all 4 of the memory planes that were included in the comparison had the same value as their memory plane's respective bits in this register.
2	**Reserved.** Read as 0.
1:0	**Write Mode.** 00 = Write Mode 0 — During a CPU write to the frame buffer, the addressed byte in each of the 4 memory planes is written with the CPU write data after it has been rotated by the number of counts specified in the Data Rotate Register (GR03). If, however, the bit(s) in the Enable Set/Reset Register (GR01) corresponding to one or more of the memory planes is set to 1, then those memory planes will be written to with the data stored in the corresponding bits in the Set/Reset Register (GR00). 01 = Write Mode 1 — During a CPU write to the frame buffer, the addressed byte in each of the 4 memory planes is written to with the data stored in the memory read latches. (The memory read latches stores an unaltered copy of the data last read from any location in the frame buffer.) 10 = Write Mode 2 — During a CPU write to the frame buffer, the least significant 4 data bits of the CPU write data is treated as the color value for the pixels in the addressed byte in all 4 memory planes. The 8 bits of the Bit Mask Register (GR08) are used to selectively enable or disable the ability to write to the corresponding bit in each of the 4 memory planes that correspond to a given pixel. A

Bit	Description
	setting of 0 in a bit in the Bit Mask Register at a given bit position causes the bits in the corresponding bit positions in the addressed byte in all 4 memory planes to be written with value of their counterparts in the memory read latches. A setting of 1 in a Bit Mask Register at a given bit position causes the bits in the corresponding bit positions in the addressed byte in all 4 memory planes to be written with the 4 bits taken from the CPU write data to thereby cause the pixel corresponding to these bits to be set to the color value. 11 = Write Mode 3 — During a CPU write to the frame buffer, the CPU write data is logically ANDed with the contents of the Bit Mask Register (GR08). The result of this ANDing is treated as the bit mask used in writing the contents of the Set/Reset Register (GR00) are written to addressed byte in all 4 memory planes.

1.3.8 GR06—Miscellaneous Register

I/O (and Memory Offset) Address: 3CFh (Index=06h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

7 4	3 2	1	0
Reserved (0000)	Memory Map Mode	Chain Odd/Even	Graphics / Text Mode

Bit	Description
7:4	**Reserved.** Read as 0s.
3:2	**Memory Map Mode.** These 2 bits control the mapping of the VGA address range for frame buffer into the CPU address space as follows: **Bit [3:2] Frame Buffer Address Range** 00 A0000h – BFFFFh 01 A0000h – AFFFFh 10 B0000h – B7FFFh 11 B8000h – BFFFFh **Note:** • This function is used in both in standard VGA modes, extended VGA modes (132 column text), and in non-VGA modes (hi-res). (132 column text modes are no longer supported). • VGA aperture memory accesses are also controlled by the PCI configuration Memory Enable bit and the RAM enable bit in the Miscellaneous Output Register (3c2/3cc). • For accesses using GR10 and GR11 to paged VGA RAM or to device MMIO registers, set these bits to 01to select the (A0000-AFFFF) range. • The CPU must map this memory as uncacheable (UC); see **VGA Host Access Memory Munging** in *Display and Overlay Functions*.

Bit	Description
1	**Chain Odd/Even.** This bit provides the ability to alter the interpretation of address bit A0, so that it may be used in selecting between the odd-numbered memory planes (planes 1 and 3) and the even-numbered memory planes (planes 0 and 2). 0 = A0 functions normally. 1 = A0 is switched with a high order address bit, in terms of how it is used in address decoding. The result is that A0 is used to determine which memory plane is being accessed (A0=0 for planes 0 and 2 and A0=1 for planes 1 and 3).
0	**Graphics/Text Mode.** This is one of two bits that are used to determine if the VGA is operating in text or graphics modes. The other bit is in AR10[0], these two bits need to be programmed in a consistent manner to achieve the proper results. 0 = Text mode. 1 = Graphics mode.

1.3.9 GR07—Color Don't Care Register

I/O (and Memory Offset) Address: 3CFh (Index=07h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

7 4	3	2	1	0
Reserved (0000)	Ignore Color Plane 3	Ignore Color Plane 2	Ignore Color Plane 1	Ignore Color Plane 0

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Ignore Color Plane [3:0].** Note that these bits have effect only when bit 3 of the Graphics Mode Register (GR05) is set to 1 to select read mode 1. 0 = The corresponding bit in the Color Compare Register (GR02) will not be included in color comparisons. 1 = The corresponding bit in the Color Compare Register (GR02) is used in color comparisons.

1.3.10 GR08—Bit Mask Register

I/O (and Memory Offset) Address: 3CFh (Index=08h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Bit Mask.** 0 = The corresponding bit in each of the 4 memory planes is written to with the corresponding bit in the memory read latches. 1 = Manipulation of the corresponding bit in each of the 4 memory planes via other mechanisms is enabled. **Notes:** • This bit mask applies to any writes to the addressed byte of any or all of the 4 memory planes, simultaneously. • This bit mask is applicable to any data written into the frame buffer by the CPU, including data that is also subject to rotation, logical functions (AND, OR, XOR), and Set/Reset. To perform a proper read-modify-write cycle into frame buffer, each byte must first be read from the frame buffer by the CPU (and this will cause it to be stored in the memory read latches), this Bit Mask Register must be set, and the new data then written into the frame buffer by the CPU.

1.3.11 GR10—Address Mapping

I/O (avoid MMIO access) Address: 3CFh (Index=10h)

Default: 00h

Attributes: R/W

This register should only be accessed using I/O operations and never be accessed through the A/B segment addressing map, I/O space register map, or direct MMIO operations.

7 4	3	2 1	0
Page Select Extension	I/O Map Enable	Paging Target	Page Mapping

Bit	Description
7:4	**Page Select Extension** **These bits form the upper bits of a 12-bit page selection value. They when combined with the GR11 <7:0> bits they define the offset into stolen memory to the 64KB page that is accessible via the VGA Memory paging mechanism.** This register provides the upper Address[27:24] bits for the access allowing for a maximum of 256MB address range. The actual range that can be used is limited to the size of the stolen graphics memory region. Addresses specified through these bits and the bits in GR11 should be legal addresses that have been limited to the size of stolen memory, accesses outside this range produce unspecified results.
3	**Reserved**
2:1	**Paging Map Target.** When paging is enabled, these bits determine the target for data cycle accesses through the VGA memory aperture. VGA Graphics pre-allocated memory is used for cases where there is no local memory and in devices that do not support local memory. Local memory is used when it exists. Memory mapped register access is only available through this mechanism in a few devices. 00 = VGA Graphics Memory (Local/pre-allocated starting at a base of local address zero) 01 = Reserved (Was Memory Mapped Registers for previous devices) 10 = Reserved (Was Video BIOS ROM Memory only for discrete graphics devices) 11 = Reserved

Bit	Description
0	**Page Mapping Enable.** This mode allows the mapping of the VGA memory address space to either VGA memory (pre-allocated or local). In previous devices it was used to map MMIO registers, or Video BIOS ROM. This allowed video BIOS access to the entire memory mapped register space, allow real mode DOS applications for BIOS flash operations, and extended video mode support for DOS applications in cases where the frame buffer is greater than the 64K bytes. **Some Notes on Paging.** Once this is enabled, no VGA memory address swizzel will be performed, addresses are directly mapped to memory. The same thing was true for ROM or register accesses for devices that support that operation. A single paging register is used to map the 64KB [A0000:AFFFF] window. An internal address is generated using GR11[7:0] as the address lines [23:16] extension to the lower address lines of the access A[15:2]. When mapping is enabled, the B0000:BFFFF area is always disabled using GR06<3:2>=01. The use of addresses in the A0000-BFFFF range require that both the graphics device PCI configuration memory enable and MSR<1> be enabled. **Graphics Mode Select** (GMS). This field is used to select the amount of Main Memory that is "pre-allocated" to support the Internal Graphics device in VGA (non-linear) mode only. These 2 bits are valid only when Internal graphics is enabled. **(Paging must not step out of pre-allocated memory within main memory)** In discrete graphics devices, ROM pins are sometimes shared with other functions such as capture. Care must be taken to insure that ROM accesses do not conflict with other ongoing operations. In cases where SMM code is executing out of the A/B segment with both code and data cycles target the SMM memory, the programming of the target to memory mapped registers does not override SMM target selection. 0 = Disable (default) 1 = Enable

1.3.12 GR11—Page Selector

I/O Address (avoid MMIO access): 3CFh (Index=11h)

Default: 00h

Attributes: R/W

Bit	Description
7	**Reserved**
6:0	**Page Select.** When concatenated with the GR10<7:4> bits, selects a 64KB window within target area when Page Mapping is enabled (GR10[0]=1). This requires that the graphics device PCI configuration space memory enable, the GR06<3:2> bits to be 01 (select A0000-AFFFF only), and the MSR<1:1> bit to be set. This register provides the Address[23:17] bits for the access allowing for a 256MB address range. VGA paging of frame buffer memory is for non-VGA packed modes only and should not be enabled when using basic VGA modes. This register should only be accessed using I/O operations and never be accessed through the A000 segment addressing map or direct MMIO operations. Addresses generated via this method should be restricted to within the size of the pre-allocated (stolen) memory that is currently available.

1.3.13 GR18—Software Flags

I/O (and Memory Offset) Address: 3CFh (Index=18h)

Default: 00

Attribute: R/W

Bit	Description
7:0	**Software Flags.** Used as scratch pad space in video BIOS. These bits are separate from the bits which appear in the memory mapped IO space. They are used specifically by the SMI BIOS which does not have access to memory mapped IO at the time they are required. These register bits have no effect on H/W operation.

1.4 Attribute Controller Registers

Unlike the other sets of indexed registers, the attribute controller registers are not accessed through a scheme employing entirely separate index and data ports. I/O address 3C0h (or memory address 3C0h) is used both as the read and write for the index register, and as the write address for the data port. I/O address 3C1h (or memory address 3C1h) is the read address for the data port.

To write to the attribute controller registers, the index of the desired register must be written to I/O address 3C0h (or memory address 3C0h), and then the data is written to the very same I/O (memory) address. A flip-flop alternates with each write to I/O address 3C0h (or memory address 3C0h) to change its function from writing the index to writing the actual data, and back again. This flip-flop may be deliberately set so that I/O address 3C0h (or memory address 3C0h) is set to write to the index (which provides a way to set it to a known state) by performing a read operation from Input Status Register 1 (ST01) at I/O address 3BAh (or memory address 3BAh) or 3DAh (or memory address 3DAh), depending on whether the graphics system has been set to emulate an MDA or a CGA as per MSR[0].

To read from the attribute controller registers, the index of the desired register must be written to I/O address 3C0h (or memory address 3C0h), and then the data is read from I/O address 3C1h (or memory address 3C1h). A read operation from I/O address 3C1h (or memory address 3C1h) does not reset the flip-flop to writing to the index. Only a write to 3C0h (or memory address 3C0h) or a read from 3BAh or 3DAh (or memory address 3BAh or 3DAh), as described above, will toggle the flip-flop back to writing to the index.

1.4.1 ARX—Attribute Controller Index Register

I/O (and Memory Offset) Address: 3C0h

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

7	6	5	4	0
Reserved (00)		Video Enable	Attribute Controller Register Index	

Bit	Description
7:6	**Reserved.** Read as 0s.
5	**Video Enable.** Note that In the VGA standard, this is called the “Palette Address Source” bit. Clearing this bit will cause the VGA display data to become all 00 index values. For the default palette, this will cause a black screen. The video timing signals continue. Another control bit will turn video off and stop the data fetches. 0 = Disable. Attribute controller color registers (AR[00:0F]) can be accessed by the CPU. 1 = Enable. Attribute controller color registers (AR[00:0F]) are inaccessible by the CPU.

Bit	Description
4:0	**Attribute Controller Register Index.** These five bits are used to select any one of the attribute controller registers (AR[00:14]), to be accessed. **Note:** AR12 is referred to in the VGA standard as the Color Plane Enable Register. The words “plane,” “color plane,” “display memory plane,” and “memory map” have been all been used in IBM* literature on the VGA standard to describe the four separate regions in the frame buffer where the pixel color or attribute information is split up and stored in standard VGA planar modes. This use of multiple terms for the same subject was deemed to be confusing, therefore, AR12 is called the Memory Plane Enable Register. Attribute Controller Register Index.

1.4.2 AR[00:0F]—Palette Registers [0:F]

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=00h-0Fh)

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

7	6	5	0
Reserved		Palette Bits P[5:0]	

Bit	Description
7:6	**Reserved.** Read as 0.
5:0	**Palette Bits P[5:0].** In each of these 16 registers, these are the lower 6 of 8 bits that are used to map either text attributes or pixel color input values (for modes that use 16 colors) to the 256 possible colors available to be selected in the palette. **Note:** Bits 3 and 2 of the Color Select Register (AR14) supply bits P7 and P6 for the values contained in all 16 of these registers. Bits 1 and 0 of the Color Select Register (AR14) can also replace bits P5 and P4 for the values contained in all 16 of these registers, if bit 7 of the Mode Control Register (AR10) is set to 1.

1.4.3 AR10—Mode Control Register

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=10h)

Default: UUh (U=Undefined)

Attributes: Read/Write

7	6	5	4	3	2	1	0
Palette Bits P5, P4 Select	Pixel Width/ Clock Select	Pixel Panning Compat	Reserved (0)	Enable Blink/ Select Bkgnd Int	Enable Line Graphics Char Code	Select Display Type	Graphics/ Alpha Mode

Bit	Description
7	**Palette Bits P5, P4 Select.** 0 = P5 and P4 for each of the 16 selected colors (for modes that use 16 colors) are individually provided by bits 5 and 4 of their corresponding Palette Registers (AR[00:0F]). 1 = P5 and P4 for all 16 of the selected colors (for modes that use 16 colors) are provided by bits 1 and 0 of Color Select Register (AR14).
6	**Pixel Width/Clock Select.** 0 = Six bits of video data (translated from 4 bits via the palette) are output every dot clock. 1 = Two sets of 4 bits of data are assembled to generate 8 bits of video data which is output every other dot clock, and the Palette Registers (AR[00:0F]) are bypassed. **Note:** This bit is set to 0 for all of the standard VGA modes, except mode 13h.
5	**Pixel Panning Compatibility.** 0 = Scroll both the upper and lower screen regions horizontally as specified in the Pixel Panning Register (AR13). 1 = Scroll only the upper screen region horizontally as specified in the Pixel Panning Register (AR13). **Note:** This bit has application only when split-screen mode is being used, where the display area is divided into distinct upper and lower regions which function somewhat like separate displays.
4	**Reserved.** Read as 0.
3	**Enable Blinking/Select Background Intensity.** 0 = Disables blinking in graphics modes, and for text modes, sets bit 7 of the character attribute bytes to control background intensity, instead of blinking. 1 = Enables blinking in graphics modes and for text modes, sets bit 7 of the character attribute bytes to control blinking, instead of background intensity. **Note:** The blinking rate is derived by dividing the VSYNC signal. The Blink Rate Control field of the VGA control register defines the blinking rate.
2	**Enable Line Graphics Character Code.** 0 = Every 9th pixel of a horizontal line (i.e., the last pixel of each horizontal line of each 9-pixel wide character box) is assigned the same attributes as the background of the character of which the given pixel is a part. 1 = Every 9th pixel of a horizontal line (i.e., the last pixel of each horizontal line of each 9-pixel wide character box) is assigned the same attributes as the 8th pixel if the character of which the given pixel is a part. This setting is intended to accommodate the line-drawing characters of the PC's extended ASCII character set -- characters with an extended ASCII code in the range of B0h to DFh. **Note:** In IBM* literature describing the VGA standard, the range of extended ASCII codes that are said to include the line-drawing characters is mistakenly specified as C0h to DFh, rather than the correct range of B0h to DFh.
1	**Select Display Type.** 0 = Attribute bytes in text modes are interpreted as they would be for a color display. 1 = Attribute bytes in text modes are interpreted as they would be for a monochrome display.
0	**Graphics/Alphanumeric Mode.** This bit (along with GR06[0]) select either graphics mode or text mode. These two bits must be programmed in a consistent manner to achieve the desired results. 0 = Alphanumeric (text) mode. 1 = Graphics mode.

1.4.4 AR12—Memory Plane Enable Register

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=12h)

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

7 6	5 4	3	2	1	0
Reserved (00)	Video Status Mux	Enable Plane 3	Enable Plane 2	Enable Plane 1	Enable Plane 0

Bit	Description
7:6	**Reserved.** Read as 0.
5:4	**Video Status Mux.** These 2 bits are used to select 2 of the 8 possible palette bits (P7-P0) to be made available to be read via bits 5 and 4 of the Input Status Register 1 (ST01). The table below shows the possible choices. **Bit [5:4] ST01 Bit 5 ST01 Bit 4** 00 P2 (default) P0 (default) 01 P5 P4 10 P3 P1 11 P7 P6 These bits are typically unused by current software; they are provided for EGA compatibility.
3:0	**Enable Plane [3:0].** These 4 bits individually enable the use of each of the 4 memory planes in providing 1 of the 4 bits used in video output to select 1 of 16 possible colors from the palette to be displayed. 0 = Disable the use of the corresponding memory plane in video output to select colors, forcing the bit that the corresponding memory plane would have provided to a value of 0. 1 = Enable the use of the corresponding memory plane in video output to select colors. **Note:** AR12 is referred to in the VGA standard as the Color Plane Enable Register. The words “plane,” “color plane,” “display memory plane,” and “memory map” have been all been used in IBM™ literature on the VGA standard to describe the 4 separate regions in the frame buffer that are amongst which pixel color or attributes information is split up and stored in standard VGA planar modes. This use of multiple terms for the same subject was considered confusing; therefore, AR12 is called the Memory Plane Enable Register.

1.4.5 AR13—Horizontal Pixel Panning Register

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=13h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

7 – 4	3 – 0
Reserved (0000)	Horizontal Pixel Shift

Bit	Description
7:4	**Reserved.**
3:0	**Horizontal Pixel Shift 3-0.** This field holds a 4-bit value that selects the number of pixels by which the image is shifted horizontally to the left. This function is available in both text and graphics modes and allows for pixel panning. In text modes with a 9-pixel wide character box, the image can be shifted up to 9 pixels to the left. In text modes with an 8-pixel wide character box, and in graphics modes other than those with 256 colors, the image can be shifted up to 8 pixels to the left. A pseudo 9-bit mode is when the 9-dot character is selected but overridden by the VGA control bit. In standard VGA mode 13h (where bit 6 of the Mode Control Register, AR10, is set to 1 to support 256 colors), bit 0 of this register must remain set to 0, and the image may be shifted up to only 4 pixels to the left. In this mode, the number of pixels by which the image is shifted can be further controlled using bits 6 and 5 of the Preset Row Scan Register (CR08).

Number of Pixels Shifted				
Bits [3:0]	**9-dot**	**Pseudo 9-dot**	**8-dot**	**256-Color**
0	1	1	0	0
1	2	2	1	Undefined
2	3	3	2	1
3	4	4	3	Undefined
4	5	5	4	2
5	6	6	5	Undefined
6	7	7	6	3
7	8	7	7	Undefined
8	0	0	Undefined	Undefined

1.4.6 AR14—Color Select Register

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=14h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

7 4	3	2	1	0
Reserved (0000)	P7	P6	Alt P5	Alt P4

Bit	Description
7:4	**Reserved.**
3:2	**Palette Bits P[7:6].** These are the 2 upper-most of the 8 bits that are used to map either text attributes or pixel color input values (for modes that use 16 colors) to the 256 possible colors contained in the palette. These 2 bits are common to all 16 sets of bits P5 through P0 that are individually supplied by Palette Registers 0-F (AR[00:0F]).
1:0	**Alternate Palette Bits P[5:4].** These 2 bits can be used as an alternate version of palette bits P5 and P4. Unlike the P5 and P4 bits that are individually supplied by Palette Registers 0-F (AR[00:0F]), these 2 alternate palette bits are common to all 16 of Palette Registers. Bit 7 of the Mode Control Register (AR10) is used to select between the use of either the P5 and P4 bits that are individually supplied by the 16 Palette Registers or these 2 alternate palette bits.

1.5 VGA Color Palette Registers

In devices that have two display pipes, there are two palettes, one for each display pipe. These palettes are the same for VGA modes and non-VGA modes. Accesses through VGA register methods can optionally read or write from either one.

For each palette in the device, for each palette, the color data stored in these 256 color data positions can be accessed only through a complex sub-addressing scheme, using a data register and two index registers. The Palette Data Register at I/O address 3C9h (or memory address offset 3C1h) is the data port. The Palette Read Index Register at I/O address 3C7h (or memory address offset 3C7h) and the Palette Write Index Register at I/O address 3C8h (or memory address offset 3C8h) are the two index registers. The Palette Read Index Register is the index register that is used to choose the color data position that is to be read from via the data port, while the Palette Write Index Register is the index register that is used to choose the color data position that is to be written to through the same data port. This arrangement allows the same data port to be used for reading from and writing to two different color data positions. Reading and writing the color data at a color data position involves three successive reads or writes since the color data stored at each color data position consists of three bytes.

To read a palette color data position, the index of the desired color data position must first be written to the Palette Read Index Register. Then all three bytes of data in a given color data position may be read at the Palette Data Register. The first byte read from the Palette Data Register retrieves the 8-bit value specifying the intensity of the red color component. The second and third bytes read are the corresponding 8-bit values for the green and blue color components respectively. After completing the third read operation, the Palette Read Index Register is automatically incremented so that the data of the

next color data position becomes accessible for being read. This allows the contents of all of the 256 color data positions of the palette to be read in sequence. This is done by specifying only the index of the 0th color data position in the Palette Read Index Register, and then simply performing 768 successive reads from the Palette Data Register.

Writing a color data position, entails a very similar procedure. The index of the desired color data position must first be written to the Palette Write Index Register. Then all three bytes of data to specify a given color may be written to the Palette Data Register. The first byte written to the Palette Data Register specifies the intensity of the red color component, the second byte specifies the intensity for the green color component, and the third byte specifies the same for the blue color component. One important detail is that all three of these bytes must be written before the hardware will actually update these three values in the given color data position. When all three bytes have been written, the Palette Write Index Register is automatically incremented so that the data of the next color data position becomes accessible for being written. This allows the contents of all of the 256 color data positions of the palette to be written in sequence. This is done by specifying only the index of the 0th color data position in the Palette Write Index Register, and then simply performing 768 successive writes to the Palette Data Register.

1.5.1 DACMASK—Pixel Data Mask Register

I/O (and Memory Offset) Address: 3C6h

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Pixel Data Mask.** In indexed-color mode, the 8 bits of this register are logically ANDed with the 8 bits of pixel data received from the frame buffer for each pixel. The result of this ANDing process becomes the actual index used to select color data positions within the palette. This has the effect of limiting the choice of color data positions that may be specified by the incoming 8-bit data. 0 = Corresponding bit in the resulting 8-bit index being forced to 0. 1 = Allows the corresponding bit in the resulting index to reflect the actual value of the corresponding bit in the incoming 8-bit pixel data.

1.5.2 DACSTATE—DAC State Register

I/O (and Memory Offset) Address: 3C7h

Default: 00h

Attributes: Read Only

7 … 2	1 … 0
Reserved (000000)	DAC State

Bit	Description
7:2	**Reserved.** Read as 0.
1:0	**DAC State.** This field indicates which of the two index registers was most recently written. **Bits [1:0] Index Register Indicated** 00 Palette Write Index Register at I/O Address 3C7h (default) 01 Reserved 10 Reserved 11 Palette Read Index Register at I/O Address 3C8h

1.5.3 DACRX—Palette Read Index Register

I/O (and Memory Offset) Address: 3C7h

Default: 00h

Attributes: Write Only

Bit	Description
7:0	**Palette Read Index.** The 8-bit index value programmed into this register chooses which of 256 standard color data positions within the palette are to be made accessible for being read from via the Palette Data Register (DACDATA). The index value held in this register is automatically incremented when all three bytes of the color data position selected by the current index have been read. A write to this register will abort a uncompleted palette write sequence. This register allows access to the palette even when running non-VGA display modes.

1.5.4 DACWX—Palette Write Index Register

I/O (and Memory Offset) Address: 3C8h

Default: 00h

Attributes: Write Only

Bit	Description
7:0	**Palette Write Index.** The 8-bit index value programmed into this register chooses which of 256 standard color data positions within the palette are to be made accessible for being written via the Palette Data Register (DACDATA). The index value held in this register is automatically incremented when all three bytes of the color data position selected by the current index have been written. This register allows access to the palette even when running non-VGA display modes.

1.5.5 DACDATA—Palette Data Register

I/O (and Memory Offset) Address: 3C9h

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Palette Data.** This byte-wide data port provides read or write access to the three bytes of data of each color data position selected using the Palette Read Index Register (DACRX) or the Palette Write Index Register (DACWX). The three bytes in each color data position are read or written in three successive read or write operations. The first byte read or written specifies the intensity of the red component of the color specified in the selected color data position. The second byte is for the green component, and the third byte is for the blue component. When writing data to a color data position, all three bytes must be written before the hardware will actually update the three bytes of the selected color data position. When reading or writing to a color data position, ensure that neither the Palette Read Index Register (DACRX) or the Palette Write Index Register (DACWX) are written to before all three bytes are read or written. A write to either of these two registers causes the circuitry that automatically cycles through providing access to the bytes for red, green and blue components to be reset such that the byte for the red component is the one that will be accessed by the next read or write operation via this register. This register allows access to the palette even when running non-VGA display modes. Writes to the palette can cause sparkle if not done during inactive video periods. This sparkle is caused by an attempt to write and read the same address on the same cycle. Anti-sparkle circuits will substitute the previous pixel value for the read output.

1.6 CRT Controller Register

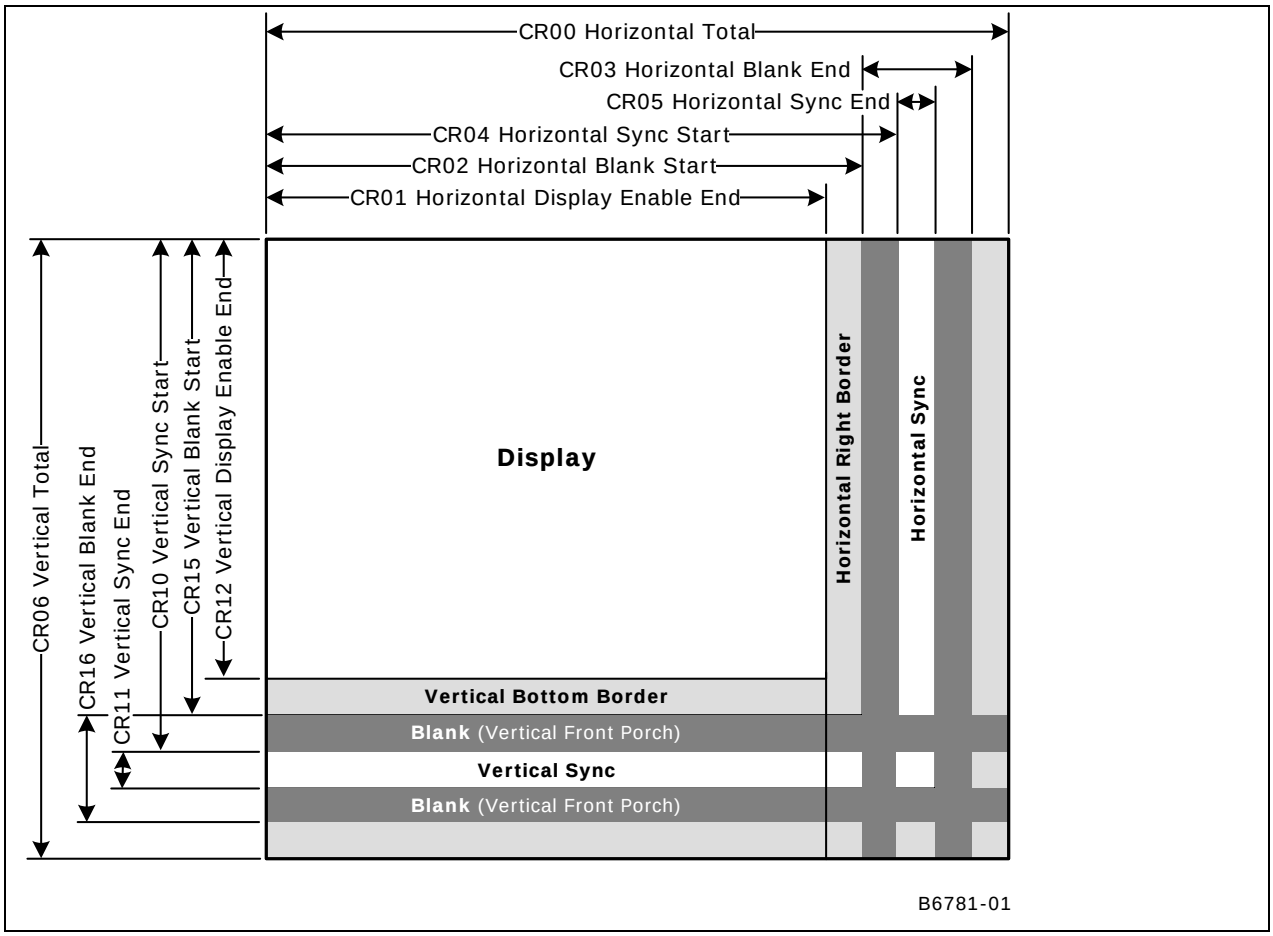
For native VGA modes, the CRTC registers determine the display timing that is to be used. In centered VGA modes, these registers determine the size of the VGA image that is to be centered in the larger timing generator defined rectangle.

The CRT controller registers are accessed by writing the index of the desired register into the CRT Controller Index Register at I/O address 3B4h or 3D4h, depending on whether the graphics system is configured for MDA or CGA emulation. The desired register is then accessed through the data port for the CRT controller registers located at I/O address 3B5h or 3D5h, again depending upon the choice of MDA or CGA emulation as per MSR[0]. For memory mapped accesses, the Index register is at 3B4h (MDA mode) or 3D3h (CGA mode) and the data port is accessed at 3B5h (MDA mode) or 3D5h (CGA mode).

Notes:

1. **Group 0 Protection:** In the original IBM VGA, CR[0:7] could be made write-protected by CR11[7]. In BIOS code, this write protection is set following each mode change. Other protection groups have no current use, and would not be used going forward by the BIOS or by drivers. They are the result of an industry fad some years ago to attempt to write protect other groups of registers; however, all such schemes were chip specific. Only the IBM compatible write protection (Group 0 Protection) is supported.

The following figure shows display fields and dimensions and the particular CRxx register that provides the control.



1.6.1 CRX—CRT Controller Index Register

I/O (and Memory Offset) Address: 3B4h/3D4h

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Reserved.** Read as 0.
6:0	**CRT Controller Index.** These 7 bits are used to select any one of the CRT controller registers to be accessed via the data port at I/O location 3B5h or 3D5h, depending upon whether the graphics system is configured for MDA or CGA emulation. The data port memory address offsets are 3B5h/3D5h.

1.6.2 CR00—Horizontal Total Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=00h)

Default: 00h

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Total.** This register is used to specify the total length of each scan line. This encompasses both the part of the scan line that is within the active display area and the part that is outside of it. Programming this register to a zero has the effect of stopping the fetching of display data. This field should be programmed with a value equal to the total number of character clocks within the entire length of a scan line, minus 5.

1.6.3 CR01—Horizontal Display Enable End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=01h)

Default: Undefined

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Display Enable End.** This register is used to specify the end of the part of the scan line that is within the active display area relative to its beginning. In other words, this is the horizontal width of the active display area. This field should be programmed with a value equal to the number of character clocks that occur within the horizontal active display area, minus 1. Horizontal display enable will go active at the beginning of each line during vertical active area, it will go inactive based on the programming of this register or the programming of the horizontal total (CR00) register. When this register value is programmed to a number that is larger than the total number of characters on a line, display enable will be active for all but the last character of the horizontal display line.

1.6.4 CR02—Horizontal Blanking Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=02h)

Default: Undefined

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Blanking Start.** This register is used to specify the beginning of the horizontal blanking period relative to the beginning of the active display area of a scan line. Horizontal blanking should always be set to start no sooner than after the end of horizontal active. This field should be programmed with a value equal to the number of character clocks that occur on a scan line from the beginning of the active display area to the beginning of the horizontal blanking.

1.6.5 CR03—Horizontal Blanking End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=03h)

Default: 1UUU UUUUb (U=Undefined)

Attributes: Read/Write (Group 0 Protection)

7	6 5	4 0
Reserved	Display Enable Skew Control	Horizontal Blanking End Bits 4:0

Bit	Description
7	**Reserved.** Values written to this bit are ignored, and to maintain consistency with the VGA standard, a value of 1 is returned when this bit is read. At one time, this bit was used to enable access to certain light pen registers. At that time, setting this bit to 0 provided this access, but setting this bit to 1 was necessary for normal operation.
6:5	**Display Enable Skew Control.** Defines the degree to which the start and end of the active display area are delayed along the length of a scan line to compensate for internal pipeline delays. These 2 bits describe the delay in terms of a number character clocks. **Bit [6:5]** — **Amount of Delay** 00 — no delay 01 — delayed by 1 character clock 10 — delayed by 2 character clocks 11 — delayed by 3 character clocks
4:0	**Horizontal Blanking End Bits [4:0].** This field provides the 5 least significant bits of a 6-bit value that specifies the end of the blanking period relative to its beginning on a single scan line. Bit 7 of the Horizontal Sync End Register (CR05) supplies the most significant bit. This 6-bit value should be programmed to be equal to the least significant 6 bits of the result of adding the length of the blanking period in terms of character clocks to the value specified in the Horizontal Blanking Start Register (CR02). End of blanking should occur before horizontal total.

1.6.6 CR04—Horizontal Sync Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=04h)

Default: Undefined

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Sync Start.** This register is used to specify the position of the beginning of the horizontal sync pulse relative to the start of the active display area on a scan line. This field should be set equal to the number of character clocks that occur from beginning of the active display area to the beginning of the horizontal sync pulse on a single scan line. Horizontal sync should always occur at least 2 clocks after the start of horizontal blank and 2 clocks before the end of horizontal blank. The actual start of sync will also be affected by both the horizontal sync skew register field and whether it is a text or graphics mode.

1.6.7 CR05—Horizontal Sync End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=05h)

Default: 00h

Attributes: Read/Write (Group 0 Protection)

7	6 5	4 0
Hor Blank End <5>	Horizontal Sync Delay	Horizontal Sync End

Bit	Description
7	**Horizontal Blanking End Bit 5.** This bit provides the most significant bit of a 6-bit value that specifies the end of the horizontal blanking period relative to its beginning. Bits [4:0] of Horizontal Blanking End Register (CR03) supplies the 5 least significant bits. See CR03[4:0] for further details. This 6-bit value should be set to the least significant 6 bits of the result of adding the length of the blanking period in terms of character clocks to the value specified in the Horizontal Blanking Start Register (CR02).
6:5	**Horizontal Sync Delay.** This field defines the degree to which the start and end of the horizontal sync pulse are delayed to compensate for internal pipeline delays. This capability is supplied to implement VGA compatibility. These field describes the delay in terms of a number character clocks. **Bit [6:5] Amount of Delay** 00 no delay 01 delayed by 1 character clock 10 delayed by 2 character clocks 11 delayed by 3 character clocks

Bit	Description
4:0	**Horizontal Sync End.** This field provides the 5 least significant bits of a 5-bit value that specifies the end of the horizontal sync pulse relative to its beginning. A value equal to the 5 least significant bits of the horizontal character counter value at which time the horizontal retrace signal becomes inactive (logical 0). Thus, this 5-bit value specifies the width of the horizontal sync pulse. To obtain a retrace signal of W, the following algorithm is used: Value of Horizontal Sync start Register (CR04) + width of horizontal retrace signal in character clock units = 5 bit result to be programmed in this field

1.6.8 CR06—Vertical Total Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=06h)

Default: 00h

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Vertical Total Bits [7:0].** This field provides the 8 least significant bits of either a 10-bit or 12-bit value that specifies the total number of scan lines. This includes the scan lines both inside and outside of the active display area. In standard VGA modes, the vertical total is specified with a 10-bit value. The 8 least significant bits of this value are supplied by these 8 bits of this register, and the 2 most significant bits are supplied by bits 5 and 0 of the Overflow Register (CR07).

1.6.9 CR07—Overflow Register (Vertical)

I/O (and Memory Offset) Address: 3B5h/3D5h (index=07h)

Default: UU0U UUU0b (U=Undefined)

Attributes: Read/Write (Group 0 Protection on bits [7:5, 3:0])

7	6	5	4	3	2	1	0
Vert Sync Start <9>	Vert Disp Enable <9>	Vert Total <9>	Line Cmp<8>	Vert Blank Start<8>	Vert Sync Start<8>	Vert Display Enable <8>	Vert Total <8>

Bit	Description
7	**Vertical Sync Start Bit 9.** The vertical sync start is a 10-bit that specifies the beginning of the vertical sync pulse relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Sync Start Register (CR10), and the most and second-most significant bits are supplied by this bit and bit 2, respectively, of this register. This 10-bit value should be programmed to be equal to the number of scan lines from the beginning of the active display area to the start of the vertical sync pulse. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical sync pulse begins.

Bit	Description
6	**Vertical Display Enable End Bit 9.** The vertical display enable end is a 10-bit that specifies the number of the last scan line within the active display area. In standard VGA modes, the vertical display enable end is specified with a 10-bit value. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Display Enable End Register (CR12), and the most and second-most significant bits are supplied by this bit and bit 1, respectively, of this register. This 10-bit value should be programmed to be equal to the number of the last scan line within in the active display area. Since the active display area always starts on the 0th scan line, this number should be equal to the total number of scan lines within the active display area, minus 1.
5	**Vertical Total Bit 9.** The vertical total is a 10-bit value that specifies the total number of scan lines. This includes the scan lines both inside and outside of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Total Register (CR06), and the most and second-most significant bits are supplied by this bit and bit 0, respectively, of this register. This 10-bit value should be programmed equal to the total number of scan lines, minus 2.
4	**Line Compare Bit 8.** This bit provides the second most significant bit of a 10-bit value that specifies the scan line at which the memory address counter restarts at the value of 0. Bit 6 of the Maximum Scan Line Register (CR09) supplies the most significant bit, and bits 7-0 of the Line Compare Register (CR18) supply the 8 least significant bits. Normally, this 10-bit value is set to specify a scan line after the last scan line of the active display area. When this 10-bit value is set to specify a scan line within the active display area, it causes that scan line and all subsequent scan lines in the active display area to display video data starting at the very first byte of the frame buffer. The result is what appears to be a screen split into a top and bottom part, with the image in the top part being repeated in the bottom part. When used in cooperation with the Start Address High Register (CR0C) and the Start Address Low Register (CR0D), it is possible to create a split display, as described earlier, but with the top and bottom parts displaying different data. The top part will display what data exists in the frame buffer starting at the address specified in the two aforementioned start address registers, while the bottom part will display what data exists in the frame buffer starting at the first byte of the frame buffer.
3	**Vertical Blanking Start Bit 8.** The vertical blanking start is a 10-bit that specifies the beginning of the vertical blanking period relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Blanking Start Register (CR15), and the most and second-most significant bits are supplied by bit 5 of the Maximum Scan Line Register (CR09) and this bit of this register, respectively. This 10-bit value should be programmed to be equal to the number of scan lines from the beginning of the active display area to the beginning of the blanking period. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical blanking period begins.
2	**Vertical Sync Start Bit 8.** The vertical sync start is a 10-bit value that specifies the beginning of the vertical sync pulse relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Sync Start Register (CR10), and the most and second-most significant bits are supplied by bit 7 and this bit, respectively, of this register. This 10-bit value should be programmed to be equal to the number of scan lines from the beginning of the active display area to the start of the vertical sync pulse. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical sync pulse begins.
1	**Vertical Display Enable End Bit 8.** The vertical display enable end is a 10-bit value that specifies the number of the last scan line within the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Display Enable End Register (CR12), and the two most significant bits are supplied by bit 6 and this bit, respectively, of this register. This 10-bit or value should be programmed to be equal to the number of the last scan line within in the active display area. Since the active display area always starts on the 0th scan line, this number should be equal to the total number of scan lines within the active display area, minus 1.

Bit	Description
0	**Vertical Total Bit 8.** The vertical total is a 10-bit value that specifies the total number of scan lines. This includes the scan lines both inside and outside of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Total Register (CR06), and the most and second-most significant bits are supplied by bit 5 and this bit, respectively, of this register. This 10-bit value should be programmed to be equal to the total number of scan lines, minus 2.

1.6.10 CR08—Preset Row Scan Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=08h)

Default: 0UUU UUUUb (U=Undefined)

Attributes: Read/Write

7	6 5	4 0
Reserved (0)	Byte Panning	Starting Row Scan Count

Bit	Description
7	**Reserved.** Read as 0s.
6:5	**Byte Panning.** This field holds a 2-bit value that selects number of bytes (up to 3) by which the image is shifted horizontally to the left on the screen. This function is available in both text and graphics modes. In text modes with a 9-pixel wide character box, the image can be shifted up to 27 pixels to the left, in increments of 9 pixels. In text modes with an 8-pixel wide character box, and in all standard VGA graphics modes, the image can be shifted up to 24 pixels to the left, in increments of 8 pixels. When the Nine dot disable bit of the VGA control register is set, the pixel shift will be equivalent to the 8-dot mode. The image can be shifted still further, in increments of individual pixels, through the use of bits [3:0] of the Horizontal Pixel Panning Register (AR13). **Number of Pixels Shifted** **Bit [6:5] — 9-Pixel Text — 8-Pixel Text & Graphics** 00 — 0 — 0 01 — 9 — 8 10 — 18 — 16 11 — 27 — 24
4:0	**Starting Row Scan Count.** This field specifies which horizontal line of pixels within the character boxes of the characters used on the top-most row of text on the display will be used as the top-most scan line. The horizontal lines of pixels of a character box are numbered from top to bottom, with the top-most line of pixels being number 0. If a horizontal line of the these character boxes other than the top-most line is specified, then the horizontal lines of the character box above the specified line of the character box will not be displayed as part of the top-most row of text characters on the display. Normally, the value specified by these 5 bits should be 0, so that all of the horizontal lines of pixels within these character boxes will be displayed in the top-most row of text, ensuring that the characters in the top-most row of text do not look as though they have been cut off at the top.

1.6.11 CR09—Maximum Scan Line Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=09h)

Default: 00h

Attributes: Read/Write

7	6	5	4 … 0
Double Scanning	Line Cmp <9>	Vert Blank Start <9>	Starting Row Scan Count

Bit	Description
7	**Double Scanning Enable.** 0 = Disable. When disabled, the clock to the row scan counter is equal to the horizontal scan rate. This is the normal setting for many of the standard VGA modes. 1 = Enable. When enabled, the clock to the row scan counter is divided by 2. This is normally used to allow CGA-compatible modes that have only 200 scan lines of active video data to be displayed as 400 scan lines (each scan line is displayed twice).
6	**Line Compare Bit 9.** This bit provides the most significant bit of a 10-bit value that specifies the scan line at which the memory address counter restarts at the value of 0. Bit 4 of the Overflow Register (CR07) supplies the second most significant bit, and bits 7-0 of the Line Compare Register (CR18) supply the 8 least significant bits. Normally, this 10-bit value is set to specify a scan line after the last scan line of the active display area. When this 10-bit value is set to specify a scan line within the active display area, it causes that scan line and all subsequent scan lines in the active display area to display video data starting at the very first byte of the frame buffer. The result is what appears to be a screen split into a top and bottom part, with the image in the top part being repeated in the bottom part. When used in cooperation with the Start Address High Register (CR0C) and the Start Address Low Register (CR0D), it is possible to create a split display, as described earlier, but with the top and bottom parts displaying different data. The top part will display whatever data exists in the frame buffer starting at the address specified in the two aforementioned start address registers, while the bottom part will display whatever data exists in the frame buffer starting at the first byte of the frame buffer.
5	**Vertical Blanking Start Bit 9.** The vertical blanking start is a 10-bit value that specifies the beginning of the vertical blanking period relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Blanking Start Register (CR15), and the most and second-most significant bits are supplied by this bit and bit 3 of the Overflow Register (CR07), respectively. This 10-bit value should be programmed to be equal to the number of scan line from the beginning of the active display area to the beginning of the blanking period. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical blanking period begins.
4:0	**Starting Row Scan Count.** This field provides all 5 bits of a 5-bit value that specifies the number of scan lines in a horizontal row of text. This value should be programmed to be equal to the number of scan lines in a horizontal row of text, minus 1.

1.6.12 CR0A—Text Cursor Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Ah)

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

7 6	5	4 0
Reserved (00)	Text Cursor Off	Text Cursor Start

Bit	Description
7:6	**Reserved.** Read as 0.
5	**Text Cursor Off.** This cursor is the text cursor that is part of the VGA standard, and should not be confused with the hardware cursor and popup which are intended to be used in extended graphics modes only. This text cursor exists only in text modes, and thus, this register is entirely ignored in graphics modes. 0 = Enables the text cursor. 1 = Disables the text cursor.
4:0	**Text Cursor Start.** This field specifies which horizontal line of pixels in a character box is to be used to display the first horizontal line of the cursor in text mode. The horizontal lines of pixels in a character box are numbered from top to bottom, with the top-most line being number 0. The value specified by these 5 bits should be the number of the first horizontal line of pixels on which the cursor is to be shown.

1.6.13 CR0B—Text Cursor End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Bh)

Default: 0UUU UUUUb (U=Undefined)

Attributes: Read/Write

7	6 5	4 0
Reserved (0)	Text Cursor Skew	Text Cursor End

Bit	Description
7	**Reserved.** Read as 0.

Bit	Description
6:5	**Text Cursor Skew.** This field specifies the degree to which the start and end of each horizontal line of pixels making up the cursor is delayed to compensate for internal pipeline delays. These 2 bits describe the delay in terms of a number character clocks. **Bit [6:5]** **Amount of Delay** 00 No delay 01 Delayed by 1 character clock 10 Delayed by 2 character clocks 11 Delayed by 3 character clocks
4:0	**Text Cursor End.** This field specifies which horizontal line of pixels in a character box is to be used to display the last horizontal line of the cursor in text mode. The horizontal lines of pixels in a character box are numbered from top to bottom, with the top-most line being number 0. The value specified by these 5 bits should be the number of the last horizontal line of pixels on which the cursor is to be shown.

1.6.14 CR0C—Start Address High Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Ch)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Start Address Bits [15:8].** This register provides either bits 15 through 8 of a 16-bit value that specifies the memory address offset from the beginning of the frame buffer at which the data to be shown in the active display area begins. (default is 0) In standard VGA modes, the start address is specified with a 16-bit value. The eight bits of this register provide the eight most significant bits of this value, while the eight bits of the Start Address Low Register (CR0D) provide the eight least significant bits.

1.6.15 CR0D—Start Address Low Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Dh)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Start Address Bits [7:0].** This register provides either bits 7 through 0 of a 16 bit value that specifies the memory address offset from the beginning of the frame buffer at which the data to be shown in the active display area begins. (default is 0) In standard VGA modes the start address is specified with a 16-bit value. The eight bits of the Start Address High Register (CR0C) provide the eight most significant bits of this value, while the eight bits of this register provide the eight least significant bits.

1.6.16 CR0E—Text Cursor Location High Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Eh)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Text Cursor Location Bits [15:8].** This field provides the 8 most significant bits of a 16-bit value that specifies the address offset from the beginning of the frame buffer at which the text cursor is located. Bit 7:0 of the Text Cursor Location Low Register (CR0F) provide the 8 least significant bits.

1.6.17 CR0F—Text Cursor Location Low Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Fh)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Text Cursor Location Bits [7:0].** This field provides the 8 least significant bits of a 16-bit value that specifies the address offset from the beginning of the frame buffer at which the text cursor is located. Bits 7:0 of the Text Cursor Location High Register (CR0E) provide the 8 most significant bits.

1.6.18 CR10—Vertical Sync Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=10h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Vertical Sync Start Bits [7:0].** This register provides the 8 least significant bits of a 10-bit that specifies the beginning of the vertical sync pulse relative to the beginning of the active display area of a screen. In standard VGA modes, this value is described in 10 bits with bits [7,2] of the Overflow Register (CR07) supplying the 2 most significant bits. This 10-bit value should equal the vertical sync start in terms of the number of scan lines from the beginning of the active display area to the beginning of the vertical sync pulse. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical sync pulse begins.

1.6.19 CR11—Vertical Sync End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=11h)

Default: 0U00 UUUUb (U=Undefined)

Attributes: Read/Write

7	6	5	4	3 … 0
Protect Regs 0:7	Reserved	Vert Int Enable	Vert Int Clear	Vertical Sync End

Bit	Description
7	**Protect Registers [0:7].** Note that the ability to write to Bit 4 of the Overflow Register (CR07) is not affected by this bit (i.e., bit 4 of the Overflow Register is always writeable). 0 = Enable writes to registers CR[00:07]. (default) 1 = Disable writes to registers CR[00:07].
6	**Reserved.** In the VGA standard, this bit was used to switch between 3 and 5 frame buffer refresh cycles during the time required to draw each horizontal line.
5	**Vertical Interrupt Enable.** This bit is reserved for compatibility only. While this bit may be written or read, it's value will have no effect. Note that the VGA does not provide an interrupt signal which would be connected to an input of the system's interrupt controller. Bit 7 of Input Status Register 0 (ST00) originally indicated the status of the vertical retrace interrupt. 0 = Enable the generation of an interrupt at the beginning of each vertical retrace period. 1 = Disable the generation of an interrupt at the beginning of each vertical retrace period.
4	**Vertical Interrupt Clear.** This is reserved for compatibility only. Note that the VGA does not provide an interrupt signal which would be connected to an input of the system's interrupt controller. 0 = Setting this bit to 0 clears a pending vertical retrace interrupt. This bit must be set back to 1 to enable the generation of another vertical retrace interrupt.
3:0	**Vertical Sync End.** This 4-bit field provides a 4-bit value that specifies the end of the vertical sync pulse relative to its beginning. This 4-bit value should be set to the least significant 4 bits of the result of adding the length of the vertical sync pulse in terms of the number of scan lines that occur within the length of the vertical sync pulse to the value that specifies the beginning of the vertical sync pulse (see the description of the Vertical Sync Start Register for more details).

1.6.20 CR12—Vertical Display Enable End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=12h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Vertical Display Enable End Bits [7:0].** This register provides the 8 least significant bits of a 10-bit value that specifies the number of the last scan line within the active display area. In standard VGA modes, this value is described in 10 bits with bits [6,1] of the Overflow Register (CR07) supplying the 2 most significant bits. This 10-bit value should be programmed to be equal to the number of the last scan line within in the active display area. Since the active display area always starts on the 0th scan line, this number should be equal to the total number of scan lines within the active display area, minus 1.

1.6.21 CR13—Offset Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=13h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Offset Bits [7:0].** This register provides either all 8 bits of an 8-bit value that specifies the number of words or DWords of frame buffer memory occupied by each horizontal row of characters. Whether this value is interpreted as the number of words or DWords is determined by the settings of the bits in the Clocking Mode Register (SR01). In standard VGA modes, the offset is described with an 8-bit value, all the bits of which are provided by this register. This 8-bit value should be programmed to be equal to either the number of words or DWords (depending on the setting of the bits in the Clocking Mode Register, SR01) of frame buffer memory that is occupied by each horizontal row of characters.

1.6.22 CR14—Underline Location Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=14h)

Default: 0UUU UUUUb (U=Undefined)

Attributes: Read/Write

7	6	5	4 … 0
Reserved (0)	Dword Mode	Count By 4	Underline Location

Bit	Description
7	**Reserved.** Read as 0.
6	**DWord Mode.** 0 = Frame buffer addresses are interpreted by the frame buffer address decoder as being either byte addresses or word addresses, depending on the setting of bit 6 of the CRT Mode Control Register (CR17). 1 = Frame buffer addresses are interpreted by the frame buffer address decoder as being DWord addresses, regardless of the setting of bit 6 of the CRT Mode Control Register (CR17). Note that this bit is used in conjunction with bits 6 and 5 of the CRT Mode Control Register (CR17) to select how frame buffer addresses from the CPU are interpreted by the frame buffer address decoder as shown below: **CR14[6] CR17[6] Addressing Mode** 0 0 Word Mode 0 1 Byte Mode 1 0 DWord Mode 1 1 DWord Mode
5	**Count By 4.** 0 = The memory address counter is incremented either every character clock or every other character clock, depending upon the setting of bit 3 of the CRT Mode Control Register. 1 = The memory address counter is incremented either every 4 character clocks or every 2 character clocks, depending upon the setting of bit 3 of the CRT Mode Control Register. . This is used in mode x13 to allow for using all four planes. Note that this bit is used in conjunction with bit 3 of the CRT Mode Control Register (CR17) to select the number of character clocks are required to cause the memory address counter to be incremented as shown, below: **CR14[5] CR17[3] Address Incrementing Interval** 0 0 every character clock 0 1 every 2 character clocks 1 0 every 4 character clocks 1 1 every 2 character clocks

Bit	Description
4:0	**Underline Location.** This field specifies which horizontal line of pixels in a character box is to be used to display a character underline in text mode. The horizontal lines of pixels in a character box are numbered from top to bottom, with the top-most line being number 0. The value specified by these 5 bits should be the number of the horizontal line on which the character underline mark is to be shown.

1.6.23 CR15—Vertical Blanking Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=15h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Vertical Blanking Start Bits [7:0].** This register provides the 8 least significant bits of a 10-bit value that specifies the beginning of the vertical blanking period relative to the beginning of the active display area of the screen. In standard VGA modes, the vertical blanking start is specified with a 10-bit value. The most and second-most significant bits of this value are supplied by bit 5 of the Maximum Scan Line Register (CR09) and bit 3 of the Overflow Register (CR07), respectively. This 10-bit value should be programmed to be equal the number of scan lines from the beginning of the active display area to the beginning of the vertical blanking period. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which vertical blanking begins.

1.6.24 CR16—Vertical Blanking End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=16h)

Default: Undefined

Attributes: Read/Write

This register provides a 8-bit value that specifies the end of the vertical blanking period relative to its beginning.

Bit	Description
7:0	**Vertical Blanking End Bits [7:0].** This 8-bit value should be set equal to the least significant 8 bits of the result of adding the length of the vertical blanking period in terms of the number of scan lines that occur within the length of the vertical blanking period to the value that specifies the beginning of the vertical blanking period (see the description of the Vertical Blanking Start Register for details).

1.6.25 CR17—CRT Mode Control

I/O (and Memory Offset) Address: 3B5h/3D5h (index=17h)

Default: 0UU0 UUUUb (U=Undefined)

Attributes: Read/Write

7	6	5	4	3	2	1	0
CRT Ctrl Reset	Word or Byte Mode	Address Wrap	Reserved (0)	Count By 2	Horizontal Retrace Select	Select Row Scan Cntr	Compat Mode Support

Bit	Description
7	**CRT Controller Reset.** This bit has no effect except in native VGA modes (non-centered). 0 = Forces horizontal and vertical sync signals to be inactive. No other registers or outputs are affected. 1 = Permits normal operation.
6	**Word Mode or Byte Mode.** 0 = The memory address counter's output bits are shifted by 1 bit position before being passed on to the frame buffer address decoder such that they are made into word-aligned addresses when bit 6 of the Underline Location Register (CR17) is set to 0. 1 = The memory address counter's output bits remain unshifted before being passed on to the frame buffer address decoder such that they remain byte-aligned addresses when bit 6 of the Underline Location Register (CR17) is set to 0. Note that this bit is used in conjunction with bits 6 and 5 of the CRT Mode Control Register (CR17) to control how frame buffer addresses from the memory address counter are interpreted by the frame buffer address decoder as shown below: **CR14[6] CR17[6] Addressing Mode** 0 0 Word Mode—Addresses from the memory address counter are shifted once to become word-aligned 0 1 Byte Mode—Addresses from the memory address counter are not shifted 1 0 DWord Mode—Addresses from the memory address counter are shifted twice to become DWord-aligned 1 1 DWord Mode—Addresses from the memory address counter are shifted twice to become DWord-aligned
5	**Address Wrap.** Note that this bit is only effective when word mode is made active by setting bit 6 in both the Underline Location Register and this register to 0. 0 = Wrap frame buffer address at 16 KB. This is used in CGA-compatible modes. 1 = No wrapping of frame buffer addresses.
4	**Reserved.** Read as 0.

Bit	Description
3	**Count By 2.** This bit is used in conjunction with bit 5 of the Underline Location Register (CR14) to select the number of character clocks are required to cause the memory address counter to be incremented. 0 = The memory address counter is incremented either every character clock or every 4 character clocks, depending upon the setting of bit 5 of the Underline Location Register. 1 = The memory address counter is incremented either every other clock. **CR14[5] CR17[3] Address Incrementing Interval** 0 0 every character clock 0 1 every 2 character clocks 1 0 every 4 character clocks 1 1 every 2 character clocks
2	**Horizontal Retrace Select.** This bit provides a way of effectively doubling the vertical resolution by allowing the vertical timing counter to be clocked by the horizontal retrace clock divided by 2 (usually, it would be undivided). 0 = The vertical timing counter is clocked by the horizontal retrace clock. 1 = The vertical timing counter is clocked by the horizontal retrace clock divided by 2.
1	**Select Row Scan Counter.** 0 = A substitution takes place, where bit 14 of the 16-bit memory address generated of the memory address counter (after the stage at which these 16 bits may have already been shifted to accommodate word or DWord addressing) is replaced with bit 1 of the row scan counter at a stage just before this address is presented to the frame buffer address decoder. 1 = No substitution takes place. See following tables.
0	**Compatibility Mode Support.** 0 = A substitution takes place, where bit 13 of the 16-bit memory address generated of the memory address counter (after the stage at which these 16 bits may have already been shifted to accommodate word or DWord addressing) is replaced with bit 0 of the row scan counter at a stage just before this address is presented to the frame buffer address decoder. 1 = No substitution takes place. See following tables.

The following tables show the possible ways in which the address bits from the memory address counter can be shifted and/or reorganized before being presented to the frame buffer address decoder. First, the address bits generated by the memory address counter are reorganized, if need be, to accommodate byte, word or DWord modes. The resulting reorganized outputs (MAOut15-MAOut0) from the memory address counter may also be further manipulated with the substitution of bits from the row scan counter (RSOut1 and RSOut0) before finally being presented to the input bits of the frame buffer address decoder (FBIn15-FBIn0).

Table 1-2. Memory Address Counter Address Bits [15:0]

	Byte Mode CR14 bit 6=0 CR17 bit 6=1 CR17 bit 5=X	Word Mode CR14 bit 6=0 CR17 bit 6=0 CR17 bit 5=1	Word Mode CR14 bit 6=0 CR17 bit 6=0 CR17 bit 5=0	DWord Mode CR14 bit 6=1 CR17 bit 6=X CR17 bit 5=X
MAOut0	0	15	13	12
MAOut1	1	0	0	13
MAOut2	2	1	1	0
MAOut3	3	2	2	1
MAOut4	4	3	3	2
MAOut5	5	4	4	3
MAOut6	6	5	5	4
MAOut7	7	6	6	5
MAOut8	8	7	7	6
MAOut9	9	8	8	7
MAOut10	10	9	9	8
MAOut11	11	10	10	9
MAOut12	12	11	11	10
MAOut13	13	12	12	11
MAOut14	14	13	13	12
MAOut15	15	14	14	13

X = Don't Care

Table 1-3. Frame Buffer Address Decoder

	CR17 bit 1=1	CR17 bit 1=1	CR17 bit 1=0	CR17 bit 1=0
	CR17 bit 0=1	CR17 bit 0=0	CR17 bit 0=1	CR17 bit 0=0
FBIn0	MAOut0	MAOut0	MAOut0	MAOut0
FBIn1	MAOut1	MAOut1	MAOut1	MAOut1
FBIn2	MAOut2	MAOut2	MAOut2	MAOut2
FBIn3	MAOut3	MAOut3	MAOut3	MAOut3
FBIn4	MAOut4	MAOut4	MAOut4	MAOut4
FBIn5	MAOut5	MAOut5	MAOut5	MAOut5
FBIn6	MAOut6	MAOut6	MAOut6	MAOut6
FBIn7	MAOut7	MAOut7	MAOut7	MAOut7
FBIn8	MAOut8	MAOut8	MAOut8	MAOut8
FBIn9	MAOut9	MAOut9	MAOut9	MAOut9
FBIn10	MAOut10	MAOut10	MAOut10	MAOut10
FBIn11	MAOut11	MAOut11	MAOut11	MAOut11
FBIn12	MAOut12	MAOut12	MAOut12	MAOut12
FBIn13	MAOut13	MAOut13	RSOut0	RSOut0
FBIn14	MAOut14	RSOut1	MAOut14	RSOut1
FBIn15	MAOut15	MAOut15	MAOut15	MAOut15

1.6.26 CR18—Line Compare Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=18h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Line Compare Bits [7:0].** This register provides the 8 least significant bits of a 10-bit value that specifies the scan line at which the memory address counter restarts at the value of 0. Bit 6 of the Maximum Scan Line Register (CR09) supplies the most significant bit, and bit 4 of the Overflow Register (CR07) supplies the second most significant bit. Normally, this 10-bit value is set to specify a scan line after the last scan line of the active display area. When this 10-bit value is set to specify a scan line within the active display area, it causes that scan line and all subsequent scan lines in the active display area to display video data starting at the very first byte of the frame buffer. The result is what appears to be a screen split into a top and bottom part, with the image in the top part being repeated in the bottom part. (This register is only used in split screening modes, and this is not a problem because split screening is not actually used for extended modes. As a result, there is no benefit to extending the existing overflow bits for higher resolutions.) When used in cooperation with the Start Address High Register (CR0C) and the Start Address Low Register (CR0D), it is possible to create a split display, as described earlier, but with the top and bottom parts displaying different data. The top part will display whatever data exists in the frame buffer starting at the address specified in the two aforementioned start address registers, while the bottom part will display whatever data exists in the frame buffer starting at the first byte of the frame buffer.

1.6.27 CR22—Memory Read Latch Data Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=22h)

Default: 00h

Attributes: Read Only

Bit	Description
7:0	**Memory Read Latch Data.** This field provides the value currently stored in 1 of the four memory read latches. Bits 1 and 0 of the Read Map Select Register (GR04) select which of the four memory read latches may be read via this register.

Revision History

Revision Number	Description	Revision Date
1.0	First 2011 OpenSource edition	May 2011

§§

Intel® OpenSource HD Graphics Programmer's Reference Manual (PRM)

Volume 3 Part 2: Display Registers – CPU Registers (SandyBridge)

For the 2011 Intel Core Processor Family

May 2011

Revision 1.0

NOTICE:

This document contains information on products in the design phase of development, and Intel reserves the right to add or remove product features at any time, with or without changes to this open source documentation.

Doc Ref #: IHD-OS-V3 Pt2 – 05 11

Creative Commons License

You are free to Share — to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

No Derivative Works. You may not alter, transform, or build upon this work.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The SandyBridge chipset family, Havendale/Auburndale chipset family, Intel® 965 Express Chipset Family, Intel® G35 Express Chipset, and Intel® 965GMx Chipset Mobile Family Graphics Controller may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel® sales office or your distributor to obtain the latest specifications and before placing your product order. I2C is a two-wire communications bus/protocol developed by Philips. SMBus is a subset of the I2C bus/protocol and was developed by Intel®. Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

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

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

Copyright © 2011, Intel Corporation. All rights reserved.

Contents

1. CPU Display Registers [DevSNB+] ... 7

1.1 1.1 Introduction and Register Summary ... 7
1.1.1 Terminology ... 7
1.1.2 Display Mode Set Sequence ... 8

2. North Shared Functions (40000h–4FFFFh) ... 11

2.1 VGA Control Registers ... 11
2.2 Sine ROM Registers ... 15
2.2.1 2.4.1 SINE_ROM—Sine ROM ... 15
2.3 Power Measurement Registers ... 16
2.3.1 DE_POWER1 – Display Engine Power Register 1 ... 16
2.3.2 DE_POWER2 – Display Engine Power Register 2 ... 17
2.4 DPFC Control Registers (43200h–433FFh) ... 17
2.4.1 DPFC_CB_BASE – DPFC Compressed Buffer Base Address ... 17
2.4.2 DPFC_CONTROL— DPFC Control ... 18
2.4.3 DPFC_RECOMP_CTL — DPFC ReComp Control ... 20
2.4.4 DPFC_STATUS — DPFC Status ... 21
2.4.5 DPFC_CPU_Fence_Offset — Y Offset CPU Fence Base to Display Buffer Base ... 22
2.5 Interrupt Control Registers ... 23
2.5.1 Display Engine Interrupt Registers Bit Definition ... 24
2.5.2 DEISR — Display Engine Interrupt Status Register ... 26
2.5.3 DEIMR — Display Engine Interrupt Mask Register ... 27
2.5.4 DEIIR — Display Engine Interrupt Identity Register ... 28
2.5.5 DEIER — Display Engine Interrupt Enable Register ... 29
2.5.6 GT Interrupt Registers Bit Definition ... 30
2.5.7 GTISR — GT Interrupt Status Register ... 32
2.5.8 GTIMR — GT Interrupt Mask Register ... 33
2.5.9 GTIIR — GT Interrupt Identity Register ... 34
2.5.10 GTIER — GT Interrupt Enable Register ... 35
2.5.11 Power Management Interrupt Registers Bit Definition [DevSNB] ... 36
2.5.12 PMISR — PM Interrupt Status Register ... 37
2.5.13 PMIMR — Power Management Interrupt Mask Register ... 38
2.5.14 PMIIR — Power Management Interrupt Identity Register ... 39
2.5.15 PMIER — Power Management Interrupt Enable Register ... 40
2.5.16 Digital Port Hot Plug Control Register ... 41
2.5.17 GTT Fault Status Register ... 42
2.6 Display Engine Render Response ... 44
2.6.1 Display Engine Render Response Message Bit Definition ... 44
2.6.2 DERRMR — Display Engine Render Response Mask Register ... 46
2.7 Display Arbitration Control ... 47
2.7.1 DISP_ARB_CTL—Display Arbiter Control ... 47
2.7.2 DISP_ARB_CTL2—Display Arbiter Control 2 [DevSNB] ... 49
2.8 Display Watermark Registers ... 50
2.8.1 WM0_PIPE_A—Pipe A Main Watermarks ... 51
2.8.2 WM0_PIPE_B—Pipe B Main Watermarks ... 51
2.8.3 WM1—Low Power 1 Display Watermarks ... 52
2.8.4 WM2—Low Power 2 Display Watermarks ... 53
2.8.5 WM3—Low Power 3 Display Watermarks ... 54
2.8.6 WM1S—Low Power 1 Sprite Watermark ... 55
2.9 Backlight Control and Modulation Histogram Registers ... 56

2.9.1 BLC_PWM_CTL2—Backlight PWM Control Register 2 56
2.9.2 BLC_PWM_CTL—Backlight PWM Control Register 57
2.9.3 BLM_HIST_CTL—Image Enhancement Histogram Control Register 58
2.9.4 Image Enhancement Bin Data Register 60
2.9.5 Histogram Threshold Guardband Register 61
2.10 Motion Blur Mitigation (MBM) Control 62
2.10.1 MBM_CTRL—MBM Control 62
2.10.2 MBM_TBL—MBM Overdrive Table 63
2.11 Color Conversion and Control Registers 64
2.11.1 Pipe A Color Control 67
2.11.2 Pipe B Color Control 75
2.12 Display Palette Registers (4A000h–4CFFFh) 83
2.12.1 LGC_PALETTE_A—Pipe A Legacy Display Palette 84
2.12.2 LGC_PALETTE_B—Pipe B Legacy Display Palette 85
2.12.3 PREC_PALETTE_A—Pipe A Precision Display Palette 85
2.12.4 PREC_PALETTE_B—Pipe B Precision Display Palette 87
2.12.5 PIPEAGCMAX—Pipe A Gamma Correction Max 89
2.12.6 PIPEBGCMAX —Pipe B Gamma Correction Max 90
2.13 Software Flag Registers (4F000h–4F10Fh) 90
2.13.1 Software Flag Registers 90
2.13.2 DE_LOAD_SL — Display Load Scan Lines 91
3. North Pipe and Port Controls (60000h–6FFFFh) 93
3.1 Pipe A Timing 93
3.1.1 HTOTAL_A—Pipe A Horizontal Total Register 93
3.1.2 HBLANK_A—Pipe A Horizontal Blank Register 94
3.1.3 HSYNC_A—Pipe A Horizontal Sync Register 95
3.1.4 VTOTAL_A—Pipe A Vertical Total Register 96
3.1.5 VBLANK_A—Pipe A Vertical Blank Register 97
3.1.6 VSYNC_A—Pipe A Vertical Sync Register 98
3.1.7 PIPEASRC—Pipe A Source Image Size 99
3.1.8 VSYNCSHIFT_A— Vertical Sync Shift Register 100
3.1.9 Pipe A M/N Values 100
3.2 Pipe B Timing 109
3.2.1 HTOTAL_B—Pipe B Horizontal Total Register 109
3.2.2 HBLANK_B—Pipe B Horizontal Blank Register 109
3.2.3 HSYNC_B—Pipe B Horizontal Sync Register 110
3.2.4 VTOTAL_B—Pipe B Vertical Total Register 110
3.2.5 VBLANK_B—Pipe B Vertical Blank Register 111
3.2.6 VSYNC_B—Pipe B Vertical Sync Register 111
3.2.7 PIPEBSRC—Pipe B Source Image Size 112
3.2.8 VSYNCSHIFT_B— Vertical Sync Shift Register 112
3.3 Pipe B M/N Values 113
3.3.1 PipeBDataM1— Pipe B Data M value 1 113
3.3.2 PipeBDataN1— Pipe B Data N value 1 113
3.3.3 PipeBDataM2— Pipe B Data M value 2 114
3.3.4 PipeBDataN2— Pipe B Data N value 2 114
3.3.5 PipeBDPLinkM1— Pipe B Link M value 1 115
3.3.6 PipeBDPLinkN1— Pipe B Link N value 1 115
3.3.7 PipeBDPLinkM2— Pipe B Link M value 2 116
3.3.8 PipeBDPLinkN2— Pipe B Link N value 2 116
3.4 Embedded DP CTL & Aux Channel 117
3.4.1 DP_A—DisplayPort A Control Register 117
3.4.2 DPA_AUX_CH_CTL—Display Port A AUX Channel Control 121

3.4.3 DPA_AUX_CH_DATA1—Display Port A AUX Data Register 1 123
3.4.4 DPA_AUX_CH_DATA2—Display Port A AUX Data Register 2 123
3.4.5 DPA_AUX_CH_DATA3—Display Port A AUX Data Register 3 124
3.4.6 DPA_AUX_CH_DATA4—Display Port A AUX Data Register 4 124
3.4.7 DPA_AUX_CH_DATA5—Display Port A AUX Data Register 5 125
3.5 Panel Fitter Control Registers 125
3.5.1 PF_WIN_POS—Panel Fitter Window Position 125
3.5.2 PF_WIN_SZ—Panel Fitter Window Size 126
3.5.3 PF_CTRL_1—Panel Fitter Control 1 127
3.5.4 Panel Fitter Coefficient Registers 130
3.6 Panel Fitter Horizontal Coefficients 131
3.6.1 PF_HFILTL_COEF—Panel Fitter Horizontal Luma/Red Coefficients 132
3.6.2 PF_HFILTC_COEF—Panel Fitter Horizontal Chroma/Green and Blue Coefficients 133
3.7 Panel Fitter Vertical Coefficients 134
3.7.1 PF_VFILTL_COEF—Panel Fitter Vertical Luma/Red Coefficients 134
3.7.2 PF_VFILTC_COEF—Panel Fitter Vertical Chroma/Green and Blue Coefficients 135
4. Plane Controls (70000h–7FFFFh) 137
4.1 Display Pipeline A 137
4.1.1 PIPEA_DSL—Pipe A Display Scan Line 137
4.1.2 PIPEA_SLC—Pipe A Display Scan Line Count Range Compare 138
4.1.3 PIPEACONF—Pipe A Configuration Register 139
4.2 Display Pipeline A Counters and Timestamps 144
4.2.1 PIPEA_FRMCOUNT—Pipe A Frame Counter 144
4.2.2 PIPEA_FLIPCOUNT—Pipe A Flip Counter 144
4.2.3 PIPEA_FRMTIMESTAMP—Pipe A Frame Time Stamp 145
4.2.4 PIPEA_FLIPTIMESTAMP—Pipe A Flip Time Stamp 145
4.3 Display Timestamp 145
4.3.1 TIMESTAMP_HI—Time Stamp High Value 146
4.4 Display Pipeline B 146
4.4.1 PIPEB_DSL—Pipe B Display Scan Line 146
4.4.2 PIPEB_SLC—Pipe B Display Scan Line Count Range Compare 147
4.4.3 PIPEBCONF—Pipe B Configuration Register 148
4.5 Display Pipeline B Counters and Timestamps 153
4.5.1 PIPEB_FRMCOUNT—Pipe B Frame Counter 153
4.5.2 PIPEB_FLIPCOUNT—Pipe B Flip Counter 154
4.5.3 PIPEB_FRMTIMESTAMP—Pipe B Frame Time Stamp 154
4.5.4 PIPEB_FLIPTIMESTAMP—Pipe B Flip Time Stamp 154
4.6 Cursor A Plane Control Registers 155
4.6.1 CURACNTR—Cursor A Control Register 155
4.6.2 CURABASE—Cursor A Base Address Register 158
4.6.3 CURAPOS—Cursor A Position Register 159
4.6.4 CURAVGAPOPUPBASE—Cursor A VGA Popup Base Address Register 160
4.6.5 CURAPALET—Cursor A Palette registers 161
4.6.6 CURASURFLIVE—Cursor A Live Surface Base Address 162
4.7 Cursor B Plane Control Registers 162
4.7.1 CURBCNTR—Cursor B Control Register 163
4.7.2 CURBBASE—Cursor B Base Address Register 164
4.7.3 CURBPOS—Cursor B Position Register 165
4.7.4 CURBPALET—Cursor B Palette registers 166
4.7.5 CURBSURFLIVE—Cursor B Live Surface Base Address Register 166
4.8 Primary A Plane Control 167
4.8.2 DSPALINOFF—Primary A Linear Offset Register 170
4.8.3 DSPASTRIDE—Primary A Stride Register 171

4.8.4 DSPASURF—Primary A Surface Base Address Register 171
4.8.5 DSPATILEOFF—Primary A Tiled Offset Register 172
4.8.6 DSPASURFLIVE—Primary A Live Surface Base Address 173
4.9 Primary B Plane Control 174
4.9.1 DSPBCNTR—Primary B Control Register 174
4.9.2 DSPBLINOFF—Primary B Linear Offset Register 177
4.9.3 DSPBSTRIDE—Primary B Stride Register 177
4.9.4 DSPBSURF—Primary B Surface Base Address Register 178
4.9.5 DSPBTILEOFF—Primary B Tiled Offset Register 179
4.9.6 DSPBSURFLIVE—Primary B Live Surface Base Address 180
4.10 Video Sprite A Control 180
4.10.1 DVSACNTR—Video Sprite A Control Register 181
4.10.2 DVSALINOFF—Video Sprite A Linear Offset Register 186
4.10.3 DVSASTRIDE—Video Sprite A Stride Register 186
4.10.4 DVSAPOS—Video Sprite A Position Register 187
4.10.5 DVSASIZE—Video Sprite A Size Register 188
4.10.6 DVSAKEYVAL—Video Sprite A Color Key Value Register 189
4.10.7 DVSAKEYMSK—Video Sprite A Color Key Mask Register 190
4.10.8 DVSASURF—Video Sprite A Surface Address Register 191
4.10.9 DVSAKEYMAXVAL—Video Sprite A Color Key Max Value Register 192
4.10.10 DVSATILEOFF—Video Sprite A Tiled Offset Register 193
4.10.11 DVSASURFLIVE—Video Sprite A Live Surface Base Address Register 194
4.10.12 DVSASCALE—Video Sprite A Scaler Control 195
4.10.13 DVSAGAMC—Video Sprite A Gamma Correction Registers 197
4.11 Video Sprite B Control 199
4.11.1 DVSBCNTR—Video Sprite B Control Register 200
4.11.2 DVSBLINOFF—Video Sprite B Linear Offset Register 205
4.11.3 DVSBSTRIDE—Video Sprite B Stride Register 206
4.11.4 DVSBPOS—Video Sprite B Position Register 206
4.11.5 DVSBSIZE—Video Sprite B Size Register 207
4.11.6 DVSBKEYVAL—Video Sprite B Color Key Value Register 208
4.11.7 DVSBKEYMSK—Video Sprite B Color Key Mask Register 209
4.11.8 DVSBSURF—Video Sprite B Surface Address Register 210
4.11.9 DVSBKEYMAXVAL—Video Sprite B Color Key Max Value Register 210
4.11.10 DVSBTILEOFF—Video Sprite B Tiled Offset Register 211
4.11.11 DVSBSURFLIVE—Video Sprite B Live Surface Base Address Register 212
4.11.12 DVSBSCALE—Video Sprite B Scaler Control 212
4.11.13 DVSBGAMC—Video Sprite B Gamma Correction Registers 214

1. CPU Display Registers [DevSNB+]

This chapter discusses CPU Display Registers that are active in SandyBridge and later projects, unless indicated otherwise.

1.1 1.1 Introduction and Register Summary

This chapter contains the register descriptions for the display portion of a family of integrated graphics devices. These registers do vary by devices within the family of devices so special attention needs to be paid to which devices use which registers and register fields.

Different devices within the family may add, modify, or delete registers or register fields relative to another device in the same family based on the supported functions of that device.

The following table contains the sections break down where the register information is contained within this chapter:

Address Range	Description
40000h-4FFFFh	Shared Functions
60000h–6FFFFh	Pipe and Port Controls
70000h–7FFFFh	Plane Controls

1.1.1 Terminology

Description	Software Use	Should be implemented as
Read/Write	This bit can be read or written.	
Reserved	Don't assume a value for these bits. Writes have no effect.	Writes are ignored. Reads return zero.
Reserved: write as zero, must be zero, MBZ	Software must always write a zero to these bits. This allows new features to be added using these bits that will be disabled when using old software and as the default case.	Writes are ignored. Reads return zero. Maybe be connected as Read/Write in future projects.
Reserved: PBC, software must preserve contents	Software must write the original value back to this bit. This allows new features to be added using these bits.	Read only or test mode Read/Write.
Read Only	This bit is read only. The read value is determined by hardware. Writes to this bit have no effect.	According to each specific bit. The bit value is determined by hardware and not affected by register writes to the actual bit.
Read/Clear	This bit can be read. Writes to it with a one cause the bit to clear.	Hardware events cause the bit to be set and the bit will be cleared on a write operation where the corresponding bit has a one for a value.

Description	Software Use	Should be implemented as
Double Buffered	Write when desired. Read gives the unbuffered value (written value) unless specified otherwise. Written values will update to take effect after a certain point. Some have a specific arming sequence where a write to another register is required before the update can take place. This is used to ensure atomic updates of several registers.	Two stages of registers used. First stage is written into and used for readback (unless specified otherwise). First stage value is transferred into second stage at the update point. Second stage value is used to control hardware. Arm/disarm flag needed for specific arming sequences.

1.1.2 Display Mode Set Sequence

Wait values
CPU DP PLL warmup = 20uS CPU FDI transmitter PLL warmup = 10us DMI latency = 20uS FDI training pattern 1 time = 0.5uS FDI training pattern 2 time = 1.5uS FDI idle pattern time = 31uS
Enable sequence
1. Enable panel power as needed to retrieve panel configuration (use AUX VDD enable bit 2. Enable PCH clock reference source and PCH SSC modulator, wait for warmup (Can be done anytime before enabling port) 3. If enabling CPU embedded DisplayPort A: (Can be done anytime before enabling CPU pipe or port) a. Enable PCH 120MHz clock source output to CPU, wait for DMI latency b. Configure and enable CPU DisplayPort PLL in the DisplayPort A register, wait for warmup 4. If enabling port on PCH: (Must be done before enabling CPU pipe or FDI) a. Enable PCH FDI Receiver PLL, wait for warmup plus DMI latency b. Switch from Rawclk to PCDclk in FDI Receiver (FDI A OR FDI B) c. [DevSNB] Enable CPU FDI Transmitter PLL, wait for warmup d. [DevILK] CPU FDI PLL is always on and does not need to be enabled 5. Enable CPU panel fitter if needed for hires, required for VGA (Can be done anytime before enabling CPU pipe) 6. Configure CPU pipe timings, M/N/TU, and other pipe settings (Can be done anytime before enabling CPU pipe) 7. Enable CPU pipe 8. Configure and enable CPU planes (VGA or hires) 9. If enabling port on PCH: a. Program PCH FDI Receiver TU size same as Transmitter TU size for TU error checking b. Train FDI i. Set pre-emphasis and voltage (iterate if training steps fail) ii. Enable CPU FDI Transmitter and PCH FDI Receiver with Training Pattern 1 enabled.

iii. Wait for FDI training pattern 1 time

iv. Read PCH FDI Receiver ISR ([DevIBX-B+] IIR) for bit lock in bit 8 (retry at least once if no lock)

v. Enable training pattern 2 on CPU FDI Transmitter and PCH FDI Receiver

vi. Wait for FDI training pattern 2 time

vii. Read PCH FDI Receiver ISR ([DevIBX-B+] IIR) for symbol lock in bit 9 (retry at least once if no lock)

viii. Enable normal pixel output on CPU FDI Transmitter and PCH FDI Receiver

ix. Wait for FDI idle pattern time for link to become active

c. Configure and enable PCH DPLL, wait for PCH DPLL warmup (Can be done anytime before enabling PCH transcoder)

d. [DevCPT] Configure DPLL SEL to set the DPLL to transcoder mapping and enable DPLL to the transcoder.

e. [DevCPT] Configure DPLL_CTL DPLL_HDMI_multipler.

f. Configure PCH transcoder timings, M/N/TU, and other transcoder settings (should match CPU settings).

g. [DevCPT] Configure and enable Transcoder DisplayPort Control if DisplayPort will be used

h. Enable PCH transcoder

10. Enable ports (DisplayPort must enable in training pattern 1)
11. Enable panel power through panel power sequencing
12. Wait for panel power sequencing to reach enabled steady state
13. Disable panel power override
14. If DisplayPort, complete link training
15. Enable panel backlight

Disable sequence

1. Disable Panel backlight
2. Disable panel power through panel power sequencing
3. Disable CPU planes (VGA or hires)
4. [DevILK-A] Disable CPU panel fitter
5. Disable CPU pipe
6. Wait for CPU pipe off status (CPU pipe config register pipe state)
7. If disabling DisplayPort on PCH, write the DisplayPort control register bit 31 to 0b.
8. Disable CPU panel fitter (Can be done anytime after CPU pipe is off)
9. If disabling CPU embedded DisplayPort A
   a. Disable port
   b. Disable CPU DisplayPort PLL in the DisplayPort A register
   c. Disable PCH 120MHz clock source output to CPU
10. Else disabling port on PCH:
   a. Disable CPU FDI Transmitter and PCH FDI Receiver
   b. Disable port
   c. Disable PCH transcoder

d. Wait for PCH transcoder off status (PCH transcoder config register transcoder state) e. [DevCPT] Disable Transcoder DisplayPort Control if DisplayPort was used f. [DevCPT] Disable Transcoder DPLL Enable bit in DPLL_SEL g. Disable PCH DPLL (Can be done anytime after PCH ports and transcoder are off) h. If no other PCH transcoder is enabled i. Switch from PCDclk to Rawclk in PCH FDI Receiver ii. [DevSNB] Disable CPU FDI Transmitter PLL iii. Disable PCH FDI Receiver PLL 11. If SSC is no longer needed, disable PCH SSC modulator 12. If clock reference no longer needed, disable PCH clock reference source
Pipe timings change
Use complete disable sequence followed by complete enable sequence with new mode programmings. Please note that pipe source size can be changed on the fly when panel fitting is enabled.
Notes
CPU FDI Transmitter should not be set to idle while PCH transcoder is enabled as this will cause PCH transcoder underflow.

2. North Shared Functions (40000h–4FFFFh)

2.1 VGA Control Registers

VGACNTRL—VGA Display Plane Control Register

Register Type: MMIO
Address Offset: 41000h
Project: All
Default Value: 00002900h
Access: R/W
Size (in bits): 32

Note: VGA requires panel fitting to be enabled.

Bit	Description
31	**VGA_Display_Disable** Project: All Default Value: 0b VGA Display Enabled This bit will disable the VGA compatible display mode. It has no effect on VGA register or A0000-BFFFF memory aperture accesses which are controlled by the PCI configuration and VGA register settings. VGA display should only be enabled if all display planes other than VGA are disabled. After enabling the VGA, most display planes need to stay disabled, only the VGA popup (cursor A) can be enabled. **Note**: The VGA SR01 screen off bit must be programmed when enabling and disabling VGA. See the VGA Registers Bspec. Firmware can temporarily override VGA display to be disabled, causing this bit to become 1b (Disable). In order to properly disable VGA this bit should be programmed to 1b even if it already reads as 1b.
30	**Reserved** Project: All Format: PBC

Value	Name	Description	Project
0b	Enable	VGA Display Enabled	All
1b	Disable	VGA Display Disabled	All

VGACNTRL—VGA Display Plane Control Register

29 — **VGA_Pipe_Select**

Project: All

Default Value: 0b

For dual pipe devices, this bit determines which pipe is to receive the VGA display data. This must be changed only when the VGA display is in the disabled state via the VGA display disable bit or during the write to enable VGA display.

Value	Name	Description	Project
0b	PipeA	Selects Assigns the VGA display to Pipe A	All
1b	PipeB	Selects Assigns the VGA display to Pipe B	All

28:27 — **Reserved** Project: All Format: PBC

26 — **VGA_Border_Enable**

Project: All

Default Value: 0b

This bit determines if the VGA border areas are included in the active display area and do or do not appear on the port output.

The border if enabled will be scaled along with the pixel data. Setting this bit allows the popup to be positioned overlapping the border area of the image.

Value	Name	Description	Project
0b	Disable	VGA Border areas are not included in the image size calculations for active area.	All
1b	Enable	VGA Border areas are enabled and passed to the display pipe for display and used in the image size calculations.	All

25 — **Reserved** Project: All Format: PBC

24 — **Pipe_Color_Space_Conversion_Enable**

Project: All

Default Value: 0b

This bit enables pipe color space conversion for the VGA pixel data. CSC mode in the pipe CSC registers must be set to match the format of the VGA pixel data.

Value	Name	Description	Project
0b	Bypass	VGA pixel data bypasses the pipe color space conversion logic	All
1b	Pass	VGA pixel data passes through the pipe color space conversion logic	All

VGACNTRL—VGA Display Plane Control Register

23

VGA_Palette_Read_Select

Project: All

Default Value: 0b

This bit only applies to dual display pipe devices and determines which palette VGA palette read accesses will occur from.

VGA palette reads are reads from I/O address 0x3c9.

Value	Name	Description	Project
0b	Palette A	VGA palette reads will access Palette A	All
1b	Palette B	VGA palette reads will access Palette B	All

22

VGA_Palette_A_Write_Disable

Project: All

Default Value: 0b

This determines which palette the VGA palette writes will have as a destination. One or both palettes can be the destination. If both are disabled, writes will not affect the palette contents.

VGA palette writes are writes to I/O address 0x3C9h.

Value	Name	Description	Project
0b	Update Palette A	VGA palette writes will update Palette A	All
1b	Not Update Palette A	VGA palette writes will not update Palette A	All

21

VGA_Palette_B_Write_Disable

Project: All

Default Value: 0b

This determines which palette the VGA palette writes will have as a destination. One or both palettes can be the destination. If both are disabled, writes will not affect the palette contents.

VGA palette writes are writes to I/O address 0x3C9h.

Value	Name	Description	Project
0b	Update Palette B	VGA palette writes will update Palette B	All
1b	Not Update Palette B	VGA palette writes will not update Palette B	All

20

Legacy_VGA_8-Bit_Palette_Enable

Project: All

Default Value: 0b

This bit only affects reads and writes to the palette through VGA I/O addresses. In the 6-bit mode, the 8-bits of data are shifted up two bits on the write (upper two bits are lost) and shifted two bits down on the read. It provides backward compatibility for original VGA programs (in it's default state) as well as VESA VBE support for 8-bit palette. It does not affect palette accesses through the palette register MMIO path.

Value	Name	Description	Project
0b	6 bit DAC	6-bit DAC	All
1b	8 bit DAC	8-bit DAC	All

VGACNTRL—VGA Display Plane Control Register

19	**Reserved** Project: All
18	**Reserved**
17:16	**Reserved** Project: All
15:12	**Reserved**
11:8	Reserved
7:6	**Blink_Duty_Cycle** Project: All Default Value: 00b Controls the VGA text mode blink duty cycle relative to the VGA cursor blink duty cycle.

Value	Name	Description	Project
00b	100%	100% Duty Cycle, Full Cursor Rate	All
01b	25%	25% Duty Cycle, ½ Cursor Rate	All
10b	50%	50% Duty Cycle, ½ Cursor Rate	All
11b	75%	75% Duty Cycle, ½ Cursor Rate	All

5:0	**VSYNC_Blink_Rate** Project: All Controls the VGA blink rate in terms of the number of VSYNCs per on/off cycle. These bits are programmed with the (VSYNCs/cycle)/2-1. The proper programming of this register is determined by the VSYNC rate that the display requires when in a VGA display mode.

2.2 Sine ROM Registers

2.2.1 2.4.1 SINE_ROM—Sine ROM

SINE_ROM - Sine ROM	
Register Type:	MMIO
Address:	42200h-42203h
Project:	All
Default Value:	00000000h
Access:	R/W Special
Size (in bits):	32

This register can be used to calculate a sine (or cosine). The intent is to enable calculation of filter coefficients. The angle is written to bits [16:6] as a 11 bit, fixed point, 0.11 value (example for setting different degrees below). Then the sine is read from bits [16:6] as a 11 bit, fixed point, 1.10 value.

DWord	Bit	Description
0	31:17	**Reserved** Project: All
	16:6	**Sine** Default Value: 0b Project: All Write the angle, read the sine **Programming Notes**: Examples of values to write:00000000000b = 0 or 360 degrees01000000000b = 90 degrees10000000000b = 180 degrees11000000000b = 270 degrees **Project**: All
	5:0	**Reserved** Project: All

2.3 Power Measurement Registers

These registers are read by the PMU to get information for use in device power estimation.

2.3.1 DE_POWER1 – Display Engine Power Register 1

DE_POWER1 – Display Engine Power Register 1	
Register Type:	MMIO
Address Offset:	42400h-42403h
Project:	DevSNB
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32

Bit	Description
31:8	**Reserved** Project: All Format: MBZ
7:4	**Transmit_Lanes_Enabled** Project: All Range 0..12 The total number of eDP & FDI lanes enabled.
3:2	**Enabled_Panel_Fitters** Project: All Range 0..2 Each enabled panel fitter consumes an additional xxmW of power.
1:0	**Enabled_DPLLs** Project: All Range 0..2 Each DPLL enabled consumes xxmW of power.

2.3.2 DE_POWER2 – Display Engine Power Register 2

DE_POWER2 – Display Engine Power Register 2	
Register Type:	MMIO
Address Offset:	**42404h-42407h**
Project:	DevSNB
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32

Bit	Description
31:0	**DE_bandwidth_counter** Project: All This counter increments on every cache line put arriving at the DE. The bandwidth is estimated by taking the difference between two reads at a known interval. The counter is only reset at boot time.

2.4 DPFC Control Registers (43200h–433FFh)

2.4.1 DPFC_CB_BASE – DPFC Compressed Buffer Base Address

DPFC_CB_BASE – DPFC Compressed Buffer Base Address	
Register Type:	MMIO
Address Offset:	43200h-43203h
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

The contents of this register can not be changed while compression is enabled.

Bit	Description
31:28	**Reserved** Project: All Format: MBZ
27:12	**Compressed_Frame_Buffer_Offset_Address** Project: All This register specifies offset of the Compressed Frame Buffer from the base of stolen memory. The buffer must be 4K byte aligned.
11:0	**Reserved** Project: All Format: MBZ

2.4.2 DPFC_CONTROL— DPFC Control

DPFC_CONTROL— DPFC Control

Register Type: MMIO
Address Offset: 43208h-4320Bh
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 32

The contents of this register can not be changed, except bit 31, while compression is enabled

Bit	Description
31	**Enable_Frame_Buffer_Compression** Project: All Default Value: 0b This bit is used to globally enable DPFC function at the next Vertical Blank start. Frame buffer compression can only be enabled after selected primary plane has been enabled for one or more vertical blanks and must be disabled before disabling the primary plane.

Value	Name	Description	Project
0b	Disable	Disable frame buffer compression	All
1b	Enable	Enable frame buffer compression	All

Bit	Description
30	**Plane_Select** Project: All Default Value: 0b

Value	Name	Description	Project
0b	Plane A	Plane A	All
1b	Plane B	Plane B	All

Bit	Description
29	**CPU_Fence_Enable** Project: All Default Value: 0b

Value	Name	Description	Project
0b	No CPU Disp Buf	Display Buffer is not in a CPU fence. No modifications are allowed from CPU to the Display Buffer	All
1b	CPU Disp Buf	Display Buffer exists in a CPU fence	All

Bit	Description
28	**Reserved** Project: All Format: MBZ
27	Reserved
26	**Reserved**

DPFC_CONTROL— DPFC Control

25

Persistent_Mode

Project: All

Default Value: 0b

Value	Name	Description	Project
0b	Non Persistent	Non Persistent Mode	All
1b	Persistent	Persistent Mode. Enable the invalid modify qualify from CS	All

24:8

Reserved

7:6

Compression_Limit

Project: All

Default Value: 0b

This register sets a minimum limit on compression. It is also used to determine the size of the compressed buffer:

Compression Ratio 1, Pixel Format 16 bpp - Not Supported

Compression Ratio 1, Pixel Format 32 bpp - Supported (CFB=FB)

Compression Ratio 1/2, Pixel Format 16 bpp - Supported (CFB=FB)

Compression Ratio 1/2, Pixel Format 32 bpp - Supported (CFB=1/2 FB)

Compression Ratio 1/4, Pixel Format 16 bpp - Supported (CFB=1/2FB)

Compression Ratio 1/4, Pixel Format 32 bpp – Supported (CFB=1/4 FB)

FB = Frame Buffer Size

CFB = Compressed Frame Buffer Size

Value	Name	Description	Project
0b	**[Default]**		
00b	1:1	1:1 compression, compressed buffer is the same size as the uncompressed buffer	All
01b	2:1	2:1 compression, compressed buffer is one half the size of the uncompressed buffer.	All
10b	4:1	4:1 compression, compressed buffer is one quarter the size of the uncompressed buffer.	All
11b	Reserved	Reserved	All

DPFC_CONTROL— DPFC Control	
5:4	**Write Back Watermark** Default Value: 0b Project: All Compressed data write back engine waits for this amount of data (per segment) to be ready before writing the data out to memory. Compression SR mode must be a 1, or SR disabled for this to take effect.

Value	Name	Description	Project
0b	**[Default]**		
00b	4 cache lines	4 cache lines	All
01b	8 cache lines	8 cache lines	All
1Xb	Reserved	Reserved	All

3:0	**CPU_Fence_Number** Project: All Default Value: 0b This field specifies the CPU visible FENCE number corresponding to the placement of the uncompressed frame buffer. **[DevSNB]** iMPH will only send the host modify message when modifications are in the fence selected in the DPFC_CONTROL_SA register CPUFNCNUM field. The fence field in the FBC Host Modification message will always be 0 and this field must be programmed to 0 to match.

2.4.3 DPFC_RECOMP_CTL — DPFC ReComp Control

DPFC_RECOMP_CTL — DPFC ReComp Control

Register Type: MMIO
Address Offset: 4320Ch-4320Fh
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 32

Bit	Description
31:28	**Reserved** Project: All Format: MBZ
27	**Enable_ReComp_Stall** Project: All Default Value: 0b

Value	Name	Description	Project
0b	Disable [Default]	Disable	All
1b	Enable	Enable	All

DPFC_RECOMP_CTL — DPFC ReComp Control	
26:16	**ReComp_Stall_Invalidation_Watermark** Project: All If this many or more invalidations occur in one frame, stop compression until the number falls below watermark, then start the recomp timer.
15:6	**Reserved** Project: All Format: MBZ
5:0	**ReCompression_Timer_Count** Project: All After invalidations fall below watermark, wait this many frames before restarting the compressor. A 0 means restart compression on the following frame.

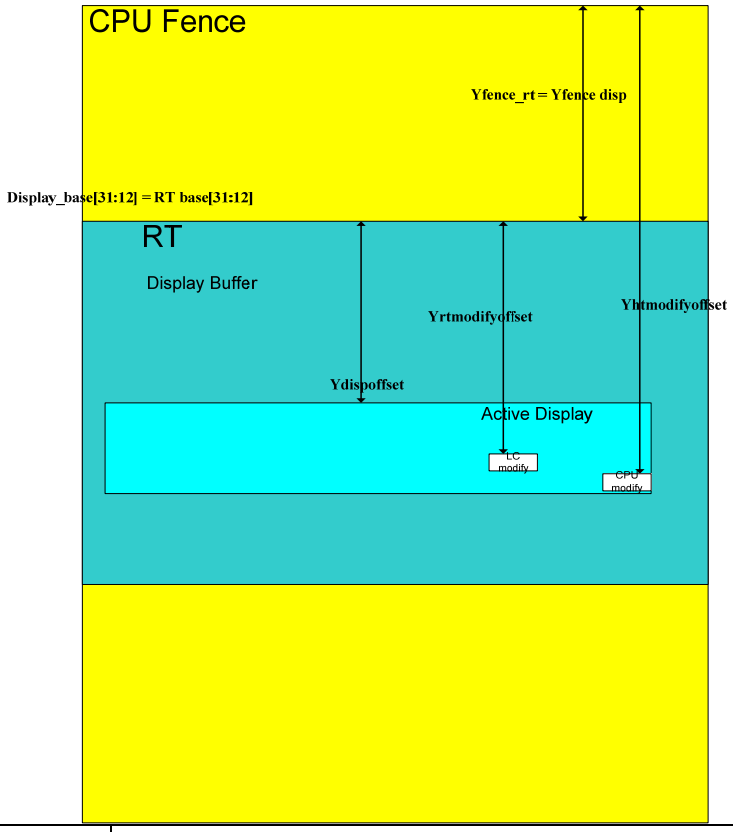
2.4.4 DPFC_STATUS — DPFC Status

DPFC_STATUS — DPFC Status	
Register Type:	MMIO
Address Offset:	43210h-43213h
Project:	All
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32
Bit	**Description**
31:27	**Reserved** Project: All Format: MBZ
26:16	**RC_Invalidated_Segment_Count** Project: [DevSNB] Updated each vblank, this field indicates the number of segments that have been invalidated for the previous frame, for RC invalidation only
15:11	**Reserved** Project: All Format: MBZ
10:0	**Compressed_Segment_Count** Project: All Updated each vblank, this field indicates the number of segments that were fetched from the compressed frame buffer for the previous frame.

2.4.5 DPFC_CPU_Fence_Offset — Y Offset CPU Fence Base to Display Buffer Base

DPFC_CPU_Fence_Offset — Y Offset CPU Fence Base to Display Buffer Base	
Register Type:	MMIO
Address Offset:	43218h-4321Bh
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

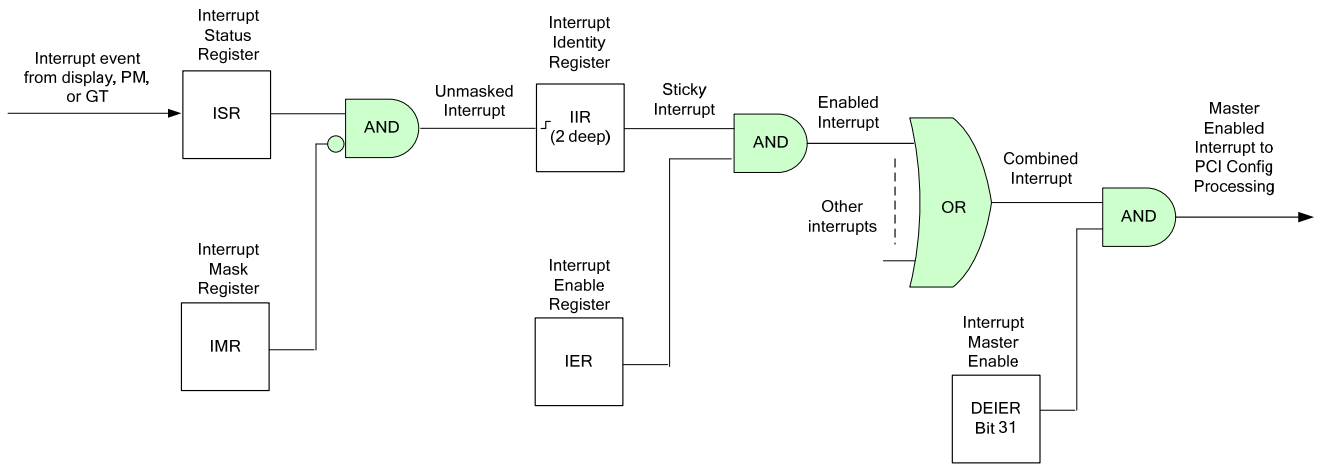
The contents of this register (shown below) can not be changed while compression is enabled.



Bit	Description
31:22	**Reserved** Project: All Format: MBZ
21:0	**Yfence_disp** Project: All Y offset from the CPU fence to the Display Buffer base. **[DevSNB]** The CPU fence is always programmed to match the Display Buffer base, so this offset must be programmed to 0 to match.

2.5 Interrupt Control Registers

First Level Interrupts in Display



For every first level interrupt bit:

Interrupt event comes in from display, PM, or GT.

There may be more levels of interrupt handling behind each event. For example the PCH Display interrupt event is the result of the SDE interrupt registers.

Interrupt event goes to Interrupt Status Register (ISR) where live status can be read back.

Interrupt event is ANDed with inverted Interrupt Mask Register (IMR) so only unmasked interrupts will proceed.

Unmasked interrupt rising edge sets sticky bit in Interrupt Indentity Register (IIR).

IIR is cleared by writing a 1b to it.

<u>IIR can queue up to two interrupt events</u>. When the IIR is cleared, it will set itself again after one clock if a second event was stored.

Sticky interrupt is ANDed with Interrupt Enable Register (IER) so only enabled interrupts will proceed.

All enabled interrupts are then ORed to create the combined interrupt.

Combined interrupt is ANDed with Master Enable (DEIER Bit 31) so only master enabled interrupt will proceed.

Master enabled interrupt then goes to PCI device 2 configuration registers PCISTS2, PCICMD2, and MC which control the MSI and line interrupt.

A Function Level Reset (FLR) or Warm Reset will reset all interrupt logic in display, causing the master enabled interrupt to deassert.

2.5.1 Display Engine Interrupt Registers Bit Definition

Display Engine Interrupt Registers Bit Definition	
Project: All **Size (in bits):** 32	
Display Engine (DE) interrupt bits come from events within the display engine, except for some which explicitly list a non-display engine source. The DEIIR and GTIIR and PMIIR are ORed together to generate the CPU interrupt. The Display Engine Interrupt Control Registers all share the same bit definitions from this table.	
Bit	**Description**
31	**Master_Interrupt_Control** Project: All Format: This bit exists only in the DEIER Display Engine Interrupt Enable Register. This is the master control for the Display to CPU interrupt. This bit must be set to 1 for any interrupts to propagate to the system.
30	**Reserved** Project: All Format:
29	**Sprite_Plane_B_flip_done** Project: All Format: This is an active high pulse when a sprite plane B flip is done.
28	**Sprite_Plane_A_flip_done** Project: All Format: This is an active high pulse when a sprite plane A flip is done.
27	**Primary_Plane_B_flip_done** Project: All Format: This is an active high pulse when a primary plane B flip is done.
26	**Primary_Plane_A_flip_done** Project: All Format: This is an active high pulse when a primary plane A flip is done.
25	**Reserved** Project: **DevSNB** Format:
24	**GTT_fault** Project: All Format: This is an active high level while either of the GTT Fault Status register bits are set.
23	**Poison** Project: All Format: This is an active high pulse on receiving the poison message.
21	**PCH_Display_interrupt_event** Project: All Format: This is an active high level while there is an interrupt being generated by the PCH Display. It will stay asserted until the interrupts in the PCH Display are all cleared. Only the rising edge of the PCH Display interrupt will cause the IIR to be set here, so all PCH Display Interrupts, including back to back interrupts, must be cleared before a new PCH Display Interrupt can cause the IIR to be set here.
20	**AUX_Channel_A** Project: All Format: This is an active high pulse on the AUX A done event.
19	**DP_A_Hotplug** Project: All Format: This is an active high level while either of the Digital Port A Hot Plug Interrupt Detect Status register bits are set.
18	**GSE** Project: DevSNB Format: This is an active high pulse on the GSE system level event.

Display Engine Interrupt Registers Bit Definition	
17	**DPST_histogram_event** Project: All Format: This is an active high pulse on the DPST histogram event.
16	**DPST_phase_in_event** Project: All Format: This is an active high pulse on the DPST phase in event.
15	**Pipe_B_vblank** Project: All Format: This is an active high level for the duration of the Pipe B vertical blank.
14	**Pipe_B_even_field** Project: All Format: This is an active high level for the duration of the Pipe B interlaced even field.
13	**Pipe_B_odd_field** Project: All Format: This is an active high level for the duration of the Pipe B interlaced odd field.
12	**Pipe_B_line_compare** Project: All Format: This is an active high level for the duration of the selected Pipe B scan lines.
11	**Pipe_B_vsync** Project: All Format: This is an active high level for the duration of the Pipe B vertical sync.
10	**Pipe_B_CRC_done** Project: All Format: This is an active high pulse on the Pipe B CRC done.
9	**Pipe_B_CRC_error** Project: All Format: This is an active high pulse on the Pipe B CRC error.
8	**Pipe_B_FIFO_underrun** Project: All Format: This is an active high level for the duration of the Pipe B FIFO underrun.
7	**Pipe_A_vblank** Project: All Format: This is an active high level for the duration of the Pipe A vertical blank.
6	**Pipe_A_even_field** Project: All Format: This is an active high level for the duration of the Pipe A interlaced even field.
5	**Pipe_A_odd_field** Project: All Format: This is an active high level for the duration of the Pipe A interlaced odd field.
4	**Pipe_A_line_compare** Project: All Format: This is an active high level for the duration of the selected Pipe A scan lines.
3	**Pipe_A_vsync** Project: All Format: This is an active high level for the duration of the Pipe A vertical sync.
2	**Pipe_A_CRC_done** Project: All Format: This is an active high pulse on the Pipe A CRC done.
1	**Pipe_A_CRC_error** Project: All Format: This is an active high pulse on the Pipe A CRC error.
0	**Pipe_A_FIFO_underrun** Project: All Format: This is an active high level for the duration of the Pipe A FIFO underrun.

2.5.2 DEISR — Display Engine Interrupt Status Register

DEISR — Display Engine Interrupt Status Register

Register Type: MMIO
Address Offset: 44000h-44003h
Project: All
Default Value: 00000000h
Access: Read Only
Size (in bits): 32

The ISR register contains the non-persistent value of all interrupt status bits. The IMR register selects which of these interrupt conditions are reported in the persistent IIR (i.e., set bits must be cleared by software). Bits in the IER are used to selectively enable IIR bits to cause CPU interrupts.

Bit	Description
31:0	**Display_Engine_ Interrupt_ Status_Bits** Project: All Format: Display Engine Interrupt Registers Bit Definition FormatDesc: Display Engine (DE) interrupt bits come from events within the display engine, except for some which explicitly list a non-display engine source. The DEIIR and GTIIR and PMIIR are ORed together to generate the CPU interrupt. The Display Engine Interrupt Control Registers all share the same bit definitions from this table.

Value	Name	Description	Project
0b	Condition Doesn't exist	Interrupt Condition currently does not exist	All
1b	Condition Exists	Interrupt Condition currently exists	All

Programming Notes
Some inputs to this register are short pulses; therefore software should not expect to use this register to sample these conditions.

2.5.3 DEIMR — Display Engine Interrupt Mask Register

DEIMR — Display Engine Interrupt Mask Register	
Register Type:	MMIO
Address Offset:	44004h-44007h
Project:	All
Default Value:	FFFFFFFFh
Access:	R/W
Size (in bits):	32

The IMR register is used by software to control which Interrupt Status Register bits are “masked” or “unmasked”. “Unmasked” bits will be reported in the IIR, possibly triggering a CPU interrupt, and will persist in the IIR until cleared by software. “Masked” bits will not be reported in the IIR and therefore cannot generate CPU interrupts.

Bit	Description
31:0	**Display Engine Interrupt Mask Bits** Project: All Format: Display Engine Interrupt Registers Bit Definition FormatDesc: Display Engine (DE) interrupt bits come from events within the display engine, except for some which explicitly list a non-display engine source. The DEIIR and GTIIR and PMIIR are ORed together to generate the CPU interrupt. The Display Engine Interrupt Control Registers all share the same bit definitions from this table. This field contains a bit mask which selects which interrupt bits from the ISR are reported in the IIR.

Value	Name	Description	Project
0b	Not Masked	Not Masked – will be reported in the IIR	All
1b	Masked	Masked – will not be reported in the IIR	All

2.5.4 DEIIR — Display Engine Interrupt Identity Register

DEIIR — Display Engine Interrupt Identity Register

Register Type: MMIO
Address Offset: 44008h
Project: All
Default Value: 00000000h
Access: R/W Clear
Size (in bits): 32

The IIR register contains the interrupt bits that are "unmasked" by the IMR and thus can generate CPU interrupts (if enabled via the IER). When a CPU interrupt is generated, this should be the first register to be interrogated to determine the source of the interrupt. **Writing a '1' into the appropriate bit position within this register clears interrupts.**

Bit	Description
31:0	**Display Engine Interrupt Identity Bits** Project: All Format: Display Engine Interrupt Registers Bit Definition FormatDesc: Display Engine (DE) interrupt bits come from events within the display engine, except for some which explicitly list a non-display engine source. The DEIIR and GTIIR and PMIIR are ORed together to generate the CPU interrupt. The Display Engine Interrupt Control Registers all share the same bit definitions from this table. This field holds the persistent values of the interrupt bits from the ISR which are unmasked by the IMR. If enabled by the IER, bits set in this register will generate a CPU interrupt. Bits set in this register will remain set (persist) until the interrupt condition is cleared via software by writing a 1' to the appropriate bit(s). For each bit, the IIR can store a second pending interrupt if two or more of the same interrupt conditions occur before the first condition is cleared. Upon clearing the interrupt, the IIR bit will momentarily go low, then return high to indicate there is another interrupt pending.

Value	Name	Description	Project
0b	Condition Not Detected	Interrupt Condition Not Detected	All
1b	Condition Detected	Interrupt Condition Detected (may or may not have actually generated a CPU interrupt)	All

2.5.5 DEIER — Display Engine Interrupt Enable Register

DEIER — Display Engine Interrupt Enable Register	
Register Type:	MMIO
Address Offset:	4400Ch
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

The IER register contains an interrupt enable bit for each interrupt bit in the IIR register. A disabled interrupt will still appear in the Interrupt Identity Register to allow polling of interrupt sources.

Bit	Description
31:0	**Display Engine Interrupt Enable Bits** Project: All Format: Display Engine Interrupt Registers Bit Definition FormatDesc: Display Engine (DE) interrupt bits come from events within the display engine, except for some which explicitly list a non-display engine source. The DEIIR and GTIIR and PMIIR are ORed together to generate the CPU interrupt. The Display Engine Interrupt Control Registers all share the same bit definitions from this table. The bits in this register enable a CPU interrupt to be generated whenever the corresponding bit in the Interrupt Identity Register becomes set. The DEIER master interrupt control bit must be set to 1 for any interrupts to propagate to the system.

Value	Name	Description	Project
0b	Disable	Disable	All
1b	Enable	Enable	All

2.5.6 GT Interrupt Registers Bit Definition

GT Interrupt Registers Bit Definition	
Project: All **Size (in bits):** 32	
The DEIIR and GTIIR and PMIIR are ORed together to generate the CPU interrupt. The GT Interrupt Control Registers all share the same bit definition from this table.	
Bit	**Description**
31	**Reserved** Project: DevSNB Format:
30	**Blitter_AS_Context_Switch_Interrupt** Project: DevSNB
29	**Blitter_page_directory_faults** Project: DevSNB
28:27	**Reserved** Project: DevSNB Format:
26	**Blitter_MI_FLUSH_DW_notify** Project: DevSNB
25	**Blitter_Command_Streamer_error_interrupt** Project: DevSNB
24	**Billter_MMIO_sync_flush_status** Project: DevSNB
23	**Reserved** Project: DevSNB Format:
22	**Blitter_Command_Streamer_MI_USER_INTERRUPT** Project: DevSNB
21	**Reserved** Project: DevSNB Format:
20	**Video_AS_Context_Switch_Interrupt** Project: DevSNB

GT Interrupt Registers Bit Definition	
19	**Video_page_directory_faults** Project: DevSNB
18	**Video_Command_Streamer_Watchdog_counter_exceeded** Project: DevSNB
17	**Reserved** Project: DevSNB Format: MBZ
16	**Video_MI_FLUSH_DW_notify** Project: DevSNB
15	**Video_Command_Streamer_error_interrupt** Project: DevSNB
14	**Video_MMIO_sync_flush_status** Project: DevSNB
13	**Reserved** Project: DevSNB Format:
12	**Video_Command_Streamer_MI_USER_INTERRUPT** Project: DevSNB
11:9	**Reserved** Project: DevSNB Format:
8	**Render_AS_Context_Switch_Interrupt** Project: DevSNB
7	**Render_page_directory_faults** Project: DevSNB
6	**Render_Command_Streamer_Watchdog_counter_exceeded** Project: DevSNB
5	**Reserved** Project: DevSNB Format:
4	**Render_PIPE_CONTROL_notify** Project: DevSNB
3	**Render_Command_Streamer_error_interrupt** Project: DevSNB
2	**Render_MMIO_sync_flush_status** Project: DevSNB

GT Interrupt Registers Bit Definition	
1	**Reserved** Project: DevSNB
0	**Render_Command_Streamer_MI_USER_INTERRUPT** Project: DevSNB

2.5.7 GTISR — GT Interrupt Status Register

GTISR — GT Interrupt Status Register

Register Type: MMIO
Address Offset: 44010h
Project: All
Default Value: 00000000h
Access: Read Only
Size (in bits): 32

The ISR register contains the non-persistent value of all interrupt status bits. The IMR register selects which of these interrupt conditions are reported in the persistent IIR (i.e., set bits must be cleared by software). Bits in the IER are used to selectively enable IIR bits to cause CPU interrupts.

Bit	Description
31:0	**GT_Interrupt_Status_Bits** Project: All Format: GT Interrupt Registers Bit Definition Format Description: The DEIIR and GTIIR and PMIIR are ORed together to generate the CPU interrupt. The GT Interrupt Control Registers all share the same bit definition from this table. This field contains the non-persistent values of all interrupt status bits.

Value	Name	Description	Project
0b	Condition Doesn't Exist	Interrupt Condition currently does not exist	All
1b	Condition Exists	Interrupt Condition currently exists	All

Programming Notes
Some inputs to this register are short pulses; therefore software should not expect to use this register to sample these conditions.

2.5.8 GTIMR — GT Interrupt Mask Register

GTIMR — GT Interrupt Mask Register

Register Type: MMIO
Address Offset: 44014h
Project: All
Default Value: FFFFFFFFh
Access: R/W
Size (in bits): 32

The IMR register is used by software to control which Interrupt Status Register bits are "masked" or "unmasked". "Unmasked" bits will be reported in the IIR, possibly triggering a CPU interrupt, and will persist in the IIR until cleared by software. "Masked" bits will not be reported in the IIR and therefore cannot generate CPU interrupts.

For command streamer interrupts DO NOT use this register to mask interrupt events. Instead use the individual command streamer MASK bits.

Bit	Description
31:0	**GT Interrupt Mask Bits** Project: All Format: GT Interrupt Registers Bit Definition FormatDesc: The DEIIR and GTIIR and PMIIR are ORed together to generate the CPU interrupt. The GT Interrupt Control Registers all share the same bit definition from this table. This field contains a bit mask which selects which interrupt bits from the ISR are reported in the IIR.

Value	Name	Description	Project
0b	Not Masked	Not Masked – will be reported in the IIR	All
1b	Masked	Masked – will not be reported in the IIR	All

2.5.9 GTIIR — GT Interrupt Identity Register

GTIIR - GT Interrupt Identity Register

Register Type:	MMIO
Address:	44018h-4401Bh
Project:	All
Default Value:	00000000h
Access:	R/W Clear
Size (in bits):	32

The IIR register contains the interrupt bits that are unmasked by the IMR and thus can generate CPU interrupts (if enabled via the IER). When a CPU interrupt is generated, this should be the first register to be interrogated to determine the source of the interrupt. **Writing a 1' into the appropriate bit position within this register clears interrupts.**

DWord	Bit	Description
0	31:0	**GT Interrupt Identity Bits** Project: All Format: GT Interrupt Registers Bit Definition FormatDesc: The DEIIR and GTIIR and PMIIR are ORed together to generate the CPU interrupt. The GT Interrupt Control Registers all share the same bit definition from this table. This field holds the persistent values of the interrupt bits from the ISR which are unmasked by the IMR. If enabled by the IER, bits set in this register will generate a CPU interrupt. Bits set in this register will remain set (persist) until the interrupt condition is cleared via software by writing a 1' to the appropriate bit(s). For each bit, the IIR can store a second pending interrupt if two or more of the same interrupt conditions occur before the first condition is cleared. Upon clearing the interrupt, the IIR bit will momentarily go low, then return high to indicate there is another interrupt pending.

Value	Name	Description	Project
0b	Condition Not Detected	Interrupt Condition Not Detected	All
1b	Condition Detected	Interrupt Condition Detected (may or may not have actually generated a CPU interrupt)	All

2.5.10 GTIER — GT Interrupt Enable Register

GTIER - GT Interrupt Enable Register

Register Type:	MMIO
Address:	4401Ch-4401Fh
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

The IER register contains an interrupt enable bit for each interrupt bit in the IIR register. A disabled interrupt will still appear in the Interrupt Identity Register to allow polling of interrupt sources.

DWord	Bit	Description
0	31:0	**GT Interrupt Enable Bits** Project: All Format: GT Interrupt Registers Bit Definition FormatDesc: The DEIIR and GTIIR and PMIIR are ORed together to generate the CPU interrupt. The GT Interrupt Control Registers all share the same bit definition from this table. The bits in this register enable a CPU interrupt to be generated whenever the corresponding bit in the Interrupt Identity Register becomes set. The DEIER master interrupt control bit must be set to 1 for any interrupts to propagate to the system.

Value	Name	Description	Project
0b	Disable	Disable	All
1b	Enable	Enable	All

2.5.11 Power Management Interrupt Registers Bit Definition [DevSNB]

Power Management Interrupt Registers Bit Definition [DevSNB]	
Project: DevSNB **Size(in bits):** 32	
The DEIIR and GTIIR and PMIIR are ORed together to generate the CPU interrupt. The Power Management Interrupt Control Registers all share the same bit definition from this table.	
Bit	**Description**
31:26	**Reserved** Project: All Format:
25	**PCU_pcode2driver_mailbox_event** Project: All Format:
24	**PCU_Thermal_Event** Project: All Format:
23:7	**Reserved** Project: All Format:
6	**Render_Frequency_Downward_Timeout_During_RC6_interrupt** Project: All Format:
5	**RP_UP_threshold_interrupt** Project: All Format:
4	**RP_DOWN_threshold_interrupt** Project: All Format:
3	**Reserved** Project: All Format:
2	**Render_geyserville_UP_evaluation_interval_interrupt** Project: All Format:
1	**Render_geyserville_Down_evaluation_interval_interrupt** Project: All Format:
0	**Reserved** Project: All Format: MBZ

2.5.12 PMISR — PM Interrupt Status Register

PMISR — PM Interrupt Status Register

Register Type: MMIO
Address Offset: 44020h
Project: DevSNB
Default Value: 00000000h
Access: Read Only
Size (in bits): 32

The ISR register contains the non-persistent value of all interrupt status bits. The IMR register selects which of these interrupt conditions are reported in the persistent IIR (i.e., set bits must be cleared by software). Bits in the IER are used to selectively enable IIR bits to cause CPU interrupts.

Bit	Description
31:0	**Power Management Interrupt Status Bits** Project: All Format: Power Management Interrupt Registers Bit Definition FormatDesc: The DEIIR and GTIIR and PMIIR are ORed together to generate the CPU interrupt. The Power Management Interrupt Control Registers all share the same bit definition from this table. This field contains the non-persistent values of all interrupt status bits.

Value	Name	Description	Project
0b	Condition Does Not Exist	Interrupt Condition currently does not exist	All
1b	Condition Exists	Interrupt Condition currently exists	All

Programming Notes	Project
Some inputs to this register are short pulses, therefore software should not expect to use this register to sample these conditions.	All

2.5.13 PMIMR — Power Management Interrupt Mask Register

PMIMR — Power Management Interrupt Mask Register

Register Type:	MMIO
Address Offset:	44024h-44027h
Project:	DevSNB
Default Value:	FFFFFFFFh
Access:	R/W
Size (in bits):	32

The IMR register is used by software to control which Interrupt Status Register bits are “masked” or “unmasked”. “Unmasked” bits will be reported in the IIR, possibly triggering a CPU interrupt, and will persist in the IIR until cleared by software. “Masked” bits will not be reported in the IIR and therefore cannot generate CPU interrupts.

For power management interrupts DO NOT use this register to mask interrupt events. Instead use the individual power management MASK bits in the corresponding PMunit register space.

Bit	Description
31:0	**Power Management Interrupt Mask Bits** Project: All Format: Power Management Interrupt Registers Bit Definition FormatDesc: The DEIIR and GTIIR and PMIIR are ORed together to generate the CPU interrupt. The Power Management Interrupt Control Registers all share the same bit definition from this table. This field contains a bit mask which selects which interrupt bits from the ISR are reported in the IIR.

Value	Name	Description	Project
0b	Not Masked	Not Masked – will be reported in the IIR	All
1b	Masked	Masked – will not be reported in the IIR	All

2.5.14 PMIIR — Power Management Interrupt Identity Register

PMIIR — Power Management Interrupt Identity Register

Register Type: MMIO
Address Offset: 44028h
Project: DevSNB
Default Value: 00000000h
Access: R/W Clear
Size (in bits): 32

The IIR register contains the interrupt bits that are "unmasked" by the IMR and thus can generate CPU interrupts (if enabled via the IER). When a CPU interrupt is generated, this should be the first register to be interrogated to determine the source of the interrupt. **Writing a '1' into the appropriate bit position within this register clears interrupts**.

Bit	Description
31:0	**Power Management Interrupt Identity Bits** Project: All Format: Power Management Interrupt Registers Bit Definition FormatDesc: The DEIIR and GTIIR and PMIIR are ORed together to generate the CPU interrupt. The Power Management Interrupt Control Registers all share the same bit definition from this table. This field holds the persistent values of the interrupt bits from the ISR which are unmasked by the IMR. If enabled by the IER, bits set in this register will generate a CPU interrupt. Bits set in this register will remain set (persist) until the interrupt condition is cleared via software by writing a 1' to the appropriate bit(s). For each bit, the IIR can store a second pending interrupt if two or more of the same interrupt conditions occur before the first condition is cleared. Upon clearing the interrupt, the IIR bit will momentarily go low, then return high to indicate there is another interrupt pending.

Value	Name	Description	Project
0b	IC No Detect	Interrupt Condition Not Detected	All
1b	IC Detect	Interrupt Condition Detected (may or may not have actually generated a CPU interrupt)	All

2.5.15 PMIER — Power Management Interrupt Enable Register

PMIER — Power Management Interrupt Enable Register

Register Type: MMIO
Address Offset: **4402Ch-4402Fh**
Project: DevSNB
Default Value: 00000000h
Access: R/W
Size (in bits): 32

The IER register contains an interrupt enable bit for each interrupt bit in the IIR register. A disabled interrupt will still appear in the Interrupt Identity Register to allow polling of interrupt sources.

Bit	Description
31:0	**Power Management Interrupt Enable Bits** Project: All Format: Power Management Interrupt Registers Bit Definition FormatDesc: The DEIIR and GTIIR and PMIIR are ORed together to generate the CPU interrupt. The Power Management Interrupt Control Registers all share the same bit definition from this table. The bits in this register enable a CPU interrupt to be generated whenever the corresponding bit in the Interrupt Identity Register becomes set. The DEIER master interrupt control bit must be set to 1 for any interrupts to propagate to the system.

Value	Name	Description	Project
0b	Disable	Disable	All
1b	Enable	Enable	All

2.5.16 Digital Port Hot Plug Control Register

Digital Port Hot Plug Control Register

Register Type: MMIO
Address Offset: **44030h-44033h**
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 32

Bit	Description
31:5	**Reserved** Project: All Format:
4	**Digital_Port_A_Hot_Plug_Detect_Input_Enable** Project: All Default Value: 0b Controls the state of the HPD buffer for the digital port. The buffer state is independent of whether the port is enabled or not.
3:2	**Digital_Port_A_Hot_Plug_Short_Pulse_Duration** Project: All Default Value: 0b These bits define the duration of the pulse defined as a short pulse.
1:0	**Digital_Port_A_Hot_Plug_Interrupt_Detect_Status** Project: All Access: R/W Clear Default Value: 0b This reflects hot plug detect status on the digital port. This bit is used for either monitor hotplug/unplug or for notification of a sink event. When either a long or short pulse is detected, one of these bits will set. These bits are ORed together to go to the main ISR hotplug register bit. These are sticky bits, cleared by writing 1s to them.

Value	Name	Description	Project
0b	Disable	Buffer disabled	All
1b	Enable	Buffer enabled. Hot plugs bit reflect the electrical state of the HPD pin	All

Value	Name	Description	Project
00b	2ms	2 ms	All
01b	4.5ms	4.5 ms	All
10b	6ms	6 ms	All
11b	100ms	100 ms	All

Value	Name	Description	Project
00b	Not Detected	Digital port hot plug event not detected	All

Digital Port Hot Plug Control Register				
	1Xb	Long Pulse	Digital port long pulse hot plug event detected	All
	X1b	Short Pulse	Digital port short pulse hot plug event detected	All

2.5.17 GTT Fault Status Register

GTT Fault Status Register

Register Type: MMIO
Address Offset: **44040h-44043h**
Project: All
Default Value: 00000000h
Access: R/W Clear
Size (in bits): 32

Bit	Description
31:8	**Reserved** Project: All Format:
7	**Invalid_GTT_page_table_entry** Project: All Default Value: 0b This is a sticky bit, cleared by writing 1 to it. All the GTT Fault Status bits are ORed together to go to the main ISR GTT Fault bit.

Value	Name	Description	Project
0b	Not Detected	Event not detected	All
1b	Detected	Event detected	All

Bit	Description
6	**Invalid_page_table_entry_data** Project: All Default Value: 0b This is a sticky bit, cleared by writing 1 to it. All the GTT Fault Status bits are ORed together to go to the main ISR GTT Fault bit.

Value	Name	Description	Project
0b	Not Detected	Event not detected	All
1b	Detected	Event detected	All

GTT Fault Status Register

5

Cursor_B_GTT_Fault_Status

Project: All

Default Value: 0b

This reflects GTT fault status for this plane. This is a sticky bit, cleared by writing 1 to it. All the GTT Fault Status bits are ORed together to go to the main ISR GTT Fault bit.

Value	Name	Description	Project
0b	Not Detected	Event not detected	All
1b	Detected	Event detected	All

4

Cursor_A_GTT_Fault_Status

Project: All

Default Value: 0b

This reflects GTT fault status for this plane. This is a sticky bit, cleared by writing 1 to it. All the GTT Fault Status bits are ORed together to go to the main ISR GTT Fault bit.

Value	Name	Description	Project
0b	Not Detected	Event not detected	All
1b	Detected	Event detected	All

3

Sprite_B_GTT_Fault_Status

Project: All

Default Value: 0b

This reflects GTT fault status for this plane. This is a sticky bit, cleared by writing 1 to it. All the GTT Fault Status bits are ORed together to go to the main ISR GTT Fault bit.

Value	Name	Description	Project
0b	Not Detected	Event not detected	All
1b	Detected	Event detected	All

2

Sprite_A_GTT_Fault_Status

Project: All

Default Value: 0b

This reflects GTT fault status for this plane. This is a sticky bit, cleared by writing 1 to it. All the GTT Fault Status bits are ORed together to go to the main ISR GTT Fault bit.

Value	Name	Description	Project
0b	Not Detected	Event not detected	All
1b	Detected	Event detected	All

GTT Fault Status Register	
1	**Primary_B_GTT_Fault_Status** Project: All Default Value: 0b This reflects GTT fault status for this plane. This is a sticky bit, cleared by writing 1 to it. All the GTT Fault Status bits are ORed together to go to the main ISR GTT Fault bit.

Value	Name	Description	Project
0b	Not Detected	Event not detected	All
1b	Detected	Event detected	All

0	**Primary_A_GTT_Fault_Status** Project: All Default Value: 0b This reflects GTT fault status for this plane. This is a sticky bit, cleared by writing 1 to it. All the GTT Fault Status bits are ORed together to go to the main ISR GTT Fault bit.

Value	Name	Description	Project
0b	Not Detected	Event not detected	All
1b	Detected	Event detected	All

2.6 Display Engine Render Response

Do not cause more than one display event to be reported in the render response.

Either mask off all but one event, using the Display Engine Render Response Mask Register (DERRMR 0x44050), or never initiate more than one event.

2.6.1 Display Engine Render Response Message Bit Definition

Display Engine Render Response Message Bit Definition

Project: DevSNB
Size(in bits): 32

Display Engine (DE) render response message bits come from events within the display engine. The Display Engine Render Response Message Registers all share the same bit definitions from this table.

Bit	Description
31:14	**Reserved** Project: All Format:
13	**Pipe_B_Start_of_Horizontal_Blank_Event** Project: All Format: This event will be reported on the start of the Pipe B Horizontal Blank.
12	**Reserved** Project: All Format:

Display Engine Render Response Message Bit Definition	
11	**Pipe_B_Start_of_Vertical_Blank_Event** Project: All Format: This event will be reported on the start of the Pipe B Vertical Blank.
10	**Pipe_B_Sprite_Plane_Flip_Done_Event** Project: All Format: This event will be reported on the completion of a flip for the Pipe B Sprite Plane.
9	**Pipe_B_Primary_Plane_Flip_Done_Event** Project: All Format: This even will be reported on the completion of a flip for the Pipe B Primary Plane.
8	**Pipe_B_Scanline_Event** Project: All Format: This event will be reported on the Pipe B scan line event.
7:6	**Reserved** Project: All Format:
5	**Pipe_A_Start_of_Horizontal_Blank_Event** Project: All Format: This event will be reported on the start of the Pipe A Horizontal Blank.
4	**Reserved** Project: All Format:
3	**Pipe_A_Start_of_Vertical_Blank_Event** Project: All Format: This even will be reported on the start of the Pipe A Vertical Blank.
2	**Pipe_A_Sprite_Plane_Flip_Done_Event** Project: All Format: This event will be reported on the completion of a flip for the Pipe A Sprite Plane.
1	**Pipe_A_Primary_Plane_Flip_Done_Event** Project: All Format: This even will be reported on the completion of a flip for the Pipe A Primary Plane.
0	**Pipe_A_Scanline_Event** Project: All Format: This event will be reported on the Pipe A Display scan line event.

2.6.2 DERRMR — Display Engine Render Response Mask Register

DERRMR - Display Engine Render Response Mask Register

Register Type:	MMIO
Address:	44050h-44053h
Project:	DevSNB
Default Value:	0000FFFFh
Access:	R/W
Size (in bits):	32

This register is used by software to control which render response message bits are masked or unmasked. Unmasked bits will cause a render response message to be sent and will be reported in that message. Masked bits will not be reported and will not cause a render response message to be sent.

Unmasked events will wake render (command streamer) as they occur, so for improved power savings it is recommended to only unmask events that are required. Programming this register can be done through MMIO or a command streamer LOAD_REGISTER_IMMEDIATE command.

DWord	Bit	Description
0	31:0	**Display Engine Render Response Message Mask Bits** Project: All Format: Display Engine Render Response Message Bit Definition FormatDesc: Display Engine (DE) render response message bits come from events within the display engine. The Display Engine Render Response Message Registers all share the same bit definitions from this table. This field contains a bit mask which selects which events cause and are reported in the render response message.

Value	Name	Description	Project
0b	Not Masked	Not Masked – will be cause and be reported in the message	All
1b	Masked	Masked – will not cause or be reported in the message	All

2.7 Display Arbitration Control

2.7.1 DISP_ARB_CTL—Display Arbiter Control

DISP_ARB_CTL—Display Arbiter Control

Register Type: MMIO
Address Offset: **45000h-45003h**
Project: All
Default Value: C2240622h
Access: R/W
Size (in bits): 32
Trusted Type: 1

Bit	Description
31	**Reserved**
30	**Reserved**
29	**Reserved**
28	**Reserved** Project: All Format:
27:26	**HP_Queue_Watermark** Project: All Default Value: 00b
25:24	**LP_Write_Request_Limit** Project: All Default Value: 10b 4 The value in this register indicates the maximum number of back to back LP write requests that will be accepted from a single client before re-arbitrating.
23:20	**TLB_Request_Limit** Project: All Default Value: 0010b 2 Range: 1..15 The value in this register indicates the maximum number of TLB requests that can be made in an arbitration loop. Range 1 – 15, (default 2). Zero is not a valid programming.

Value	Name	Description	Project
00b	1	1	All
01b	2	2	All
10b	4	4 (default)	All
11b	8	8	All

DISP_ARB_CTL—Display Arbiter Control

19:16	**TLB_Request_In-Flight_Limit** Project: All Default Value: 0100b 4 Range: 1..15 The value in this register indicates the maximum number of TLB (or VTd) requests that can be in flight at any given time. Range 1 – 15, (default 4). Zero is not a valid programming.
15	**Reserved**
14:13	**Address_Swizzling_for_Tiled-Surfaces** Project: All Default Value: 00b DRAM configuration registers show if memory address swizzling is needed.

Value	Name	Description	Project
00b	No Display	No display request address swizzling	All
01b	Enable	Enable display request address bit[6] swizzling for tiled surfaces	All
1Xb	Reserved	Reserved	All

12	**Reserved** Project: All Format:
11:8	**HP_Page_Break_Limit** Project: All Default Value: 0110b 6 Range: 1..15 The value in this register represents the maximum number of page breaks allowed in a HP request chain. Range 1 – 15, (default 6). Zero is not a valid programming.
7	**Reserved** Project: All Format:
6:0	**HP_Data_Request_Limit** Project: All Default Value: 01000010b 34 Range: 1..127 The value in this register represents the maximum number of cachelines allowed in a HP request chain. Range 1 – 127, (default 34). Zero is not a valid programming.

2.7.2 DISP_ARB_CTL2—Display Arbiter Control 2 [DevSNB]

DISP_ARB_CTL2—Display Arbiter Control 2 [DevSNB]

Register Type: MMIO
Address Offset: **45004h-45007h**
Project: DevSNB
Default Value: 00000000h
Access: R/W
Size (in bits): 32

Bit	Description
31	**Reserved**
30:9	**Reserved** Project: All Format: MBZ
8	**Fetch_Timing** Project: All Default Value: 0b The value in this register is valid only when Opportunistic Fetches are enabled. The value in this register is used to specify when an opportunistic fetch can happen. For any opportunistic fetch to happen, display should not be in the process of waking the system.

Value	Name	Description	Project
0b	FE inSR	Fetch on falling edge of inSR	All
1b	Not inSR	Fetch when not inSR	All

Bit	Description
7	**Opportunistic_Fetch_Behavior** Project: All Default Value: 0h The value in this register is valid only when Opportunistic Fetches are enabled. The value in this register represents the fetch behavior when an opportunistic fetch is triggered. For any opportunistic fetch to happen, display should not be in the process of waking the system.

Value	Name	Description	Project
0h	One Burst	One Burst Only	All
1h	Fill FIFO	Fill FIFO to Top	All

Bit	Description
6	**Reserved** Project: All Format: MBZ

DISP_ARB_CTL2—Display Arbiter Control 2 [DevSNB]	
5:4	**Inflight_HP_Read_Request_Limit** Project: All Default Value: 00b The value in this register represents the maximum number of HP read request transactions that can inflight at any given time.

Value	Name	Description	Project
00b	128 HP	128 HP inflight transactions limit	All
01b	64 HP	64 HP inflight transactions limit	All
10b	32 HP	32 HP inflight transactions limit	All
11b	16 HP	16 HP inflight transactions limit	All

3:2	**Reserved** Project: All Format:
1:0	**RTID_FIFO_Watermark** Project: All Default Value: 0b The value in this register represents the watermark value for the RTID FIFO. HP transactions will start only when the FIFO level is above or equal the watermark

Value	Name	Description	Project
00b	8 RTIDs	8 RTIDs available in FIFO	All
01b	16 RTIDs	16 RTIDs available in FIFO	All
10b	32 RTIDs	32 RTIDs available in FIFO	All
11b	RESERVED	Reserved	All

2.8 Display Watermark Registers

The watermark registers are used to control the display to memory request timing. The watermarks must be programmed according to the rules provided in the “Programming Watermarks” document. The default values of the watermarks should allow the display to operate in any mode supported by the memory configuration. However, the default watermarks are not optimized for power or memory bandwidth efficiency. Watermarks must enable from the bottom up, meaning if WM2 is disabled, WM3 must also be disabled, and if WM1 is disabled, both WM2 and WM3 must also be disabled.

2.8.1 WM0_PIPE_A—Pipe A Main Watermarks

WM0_PIPE_A—Pipe A Main Watermarks	
Register Type:	MMIO
Address Offset:	**45100h-45103h**
Project:	All
Default Value:	00783818h
Access:	R/W
Size (in bits):	32

Bit	Description
31:23	**Reserved** Project: All Format:
22:16	**Pipe_A_Primary_Watermark** Project: All Number in 64Bs of data in FIFO below which the Pipe A Primary stream will generate requests to memory
15:14	**Reserved** Project: All Format:
13:8	**Pipe_A_Sprite_Watermark** Project: All Number in 64Bs of data in FIFO below which the Pipe A Sprite stream will generate requests to memory
7:5	**Reserved** Project: All Format:
4:0	**Pipe_A_Cursor_Watermark** Project: All Number in 64Bs of data in FIFO below which the Pipe A Cursor stream will generate requests to memory

2.8.2 WM0_PIPE_B—Pipe B Main Watermarks

WM0_PIPE_B—Pipe B Main Watermarks	
Register Type:	MMIO
Address Offset:	**45104h-45107h**
Project:	All
Default Value:	00783818h
Access:	R/W
Size (in bits):	32
Trusted Type:	1

Bit	Description
31:23	**Reserved** Project: All Format:
22:16	**Pipe_B_Primary_Watermark** Project: All Number in 64Bs of data in FIFO below which the Pipe B Primary stream will generate requests to memory
15:14	**Reserved** Project: All Format:

WM0_PIPE_B—Pipe B Main Watermarks	
13:8	**Pipe_B_Sprite_Watermark** Project: All Number in 64Bs of data in FIFO below which the Pipe B Sprite stream will generate requests to memory
7:5	**Reserved** Project: All Format:
4:0	**Pipe_B_Cursor_Watermark** Project: All Number in 64Bs of data in FIFO below which the Pipe B Cursor stream will generate requests to memory

2.8.3 WM1—Low Power 1 Display Watermarks

WM1—Low Power 1 Display Watermarks	
Register Type:	MMIO
Address Offset:	**45108h-4510Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
These watermark values will be used only when one pipe is enabled and no sprites are enabled (or the conditions for using the Low Power 1 Sprite Watermark are met) and the display is in LP1 state.	
Bit	**Description**
31	**Enabled** Project: All Enables LP1 watermarks
30:24	**Latency** Project: All The latency associated with the LP1 watermarks in half usecs.
23:20	**FBC_LP1_Watermark** Project: All Number of equivalent lines of the primary display for this WM
19:17	**Reserved** Project: All Format:
16:8	**LP1_Primary_Watermark** Project: All Number in 64Bs of data in FIFO below which the Primary stream will generate requests to memory.
7:6	**Reserved** Project: All Format:
5:0	**LP1_Cursor_Watermark** Project: All Number in 64Bs of data in FIFO below which the Cursor stream will generate requests to memory.

2.8.4 WM2—Low Power 2 Display Watermarks

WM2—Low Power 2 Display Watermarks	
Register Type:	MMIO
Address Offset:	**4510Ch-4510Fh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

These watermark values will be used only when one pipe is enabled and no sprites are enabled and the display is in LP2 state.

Bit	Description
31	**Enabled** Project: All Enables LP2 watermarks LP2 watermarks must be disabled when the Low Power 1 Sprite Watermark is enabled. [DevILK] This bit must be programmed to 0 when FBC is enabled and FBC watermarks are disabled.
30:24	**Latency** Project: All The latency associated with the LP2 watermarks in half usecs.
23:20	**FBC_LP2_Watermark** Project: All Number of equivalent lines of the primary display for this WM
19:17	**Reserved** Project: All Format:
16:8	**LP2_Primary_Watermark** Project: All Number in 64Bs of data in FIFO below which the Primary stream will generate requests to memory.
7:6	**Reserved** Project: All Format:
5:0	**LP2_Cursor_Watermark** Project: All Number in 64Bs of data in FIFO below which the Cursor stream will generate requests to memory.

2.8.5 WM3—Low Power 3 Display Watermarks

WM3—Low Power 3 Display Watermarks			
Register Type: MMIO **Address Offset:** **45110h-45113h** **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 32			
These watermark values will be used only when one pipe is enabled and no sprites are enabled and the display is in LP3 state.			
Bit	**Description**		
31	**Enabled** Enables LP3 watermarks	Project:	All
30:24	**Latency** The latency associated with the LP3 watermarks in half usecs.	Project:	All
23:20	**FBC_LP3_Watermark** Number of equivalent lines of the primary display for this WM	Project:	All
19:17	**Reserved** Project: All	Format:	
16:8	**LP3_Primary_Watermark** Number in 64Bs of data in FIFO below which the Primary stream will generate requests to memory.	Project:	All
7:6	**Reserved** Project: All	Format:	
5:0	**LP3_Cursor_Watermark** Number in 64Bs of data in FIFO below which the Cursor stream will generate requests to memory.	Project:	All

2.8.6 WM1S—Low Power 1 Sprite Watermark

WM1S—Low Power 1 Sprite Watermark	
Register Type:	MMIO
Address Offset:	**45120h-45123h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

This watermark will be used only when one pipe is enabled and a sprite is enabled and sprite scaling is not enabled and the display is in LP1 state.

Bit	Description
31	**Enabled** Project: All Enables LP1 Sprite watermark. This bit allows memory self refresh to be entered when sprite is enabled. The WM2 Low Power 2 Display Watermark needs to be disabled when this watermark is enabled. This bit should only be changed when sprite is not enabled or display LP1 watermarks are not enabled.
30:8	**Reserved** Project: All Format:
7:0	**LP1_Sprite_Watermark** Project: All Default Value: 0b Number in 64Bs of data in FIFO below which the Sprite stream will generate requests to memory.

2.9 Backlight Control and Modulation Histogram Registers

2.9.1 BLC_PWM_CTL2—Backlight PWM Control Register 2

BLC_PWM_CTL2—Backlight PWM Control Register 2

Register Type: MMIO
Address Offset: **48250h-48253h**
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 32

Bit	Description
31	**PWM_Enable** Project: All Default Value: 0b This bit enables the PWM counter logic

Value	Name	Description	Project
0b	Disable	PWM disabled (drives 0 always)	All
1b	Enable	PWM enabled	All

Bit	Description
30	**Reserved** Project: All Format:
29	**PWM_Pipe_assignment** Project: All Default Value: 0b This bit assigns PWM to a pipe. The PWM counter will run off of this pipe's PLL. The PWM function must be disabled in order to change the value of this field.

Value	Name	Description	Project
0b	Pipe A	Pipe A	All
1b	Pipe B	Pipe B	All

Bit	Description
28:27	**Reserved** Project: All Format:
26	**Phase-In_Interrupt_Status** Project: All Access: R/W Clear Default Value: 0b This bit will be set by hardware when a Phase-In interrupt has occurred. Software will clear this bit by writing a '1', which will reset the interrupt generation.
25	**Phase_In_Enable** Project: All Setting this bit enables a PWM phase in based on the programming of the Phase In registers below. This bit clears itself when the phase in is completed.

BLC_PWM_CTL2—Backlight PWM Control Register 2	
24	**Phase_In_Interrupt_Enable** Project: All Setting this bit enables an interrupt to be generated when the PWM phase in is completed.
23:16	**Phase_In_time_base** Project: All Default Value: 0b This field determines the number of VBLANK events that pass before one increment occurs.

Value	Name	Description	Project
0b [Default]		Invalid	All
01h-FFh		VBlank Count	All

15:8	**Phase_In_Count** Project: All This field determines the number of increment events in this phase in. Writes to this register should only occur when hardware-phase-ins are disabled. Reads to this register can occur any time, where the value in this field indicates the number of increment events remaining to fully apply a phase-in request as hardware automatically decrements this value. A value of 0 is invalid.
7:0	**Phase_In_Increment** Project: All This field indicates the amount to adjust the PWM duty cycle register on each increment event. This is a two's complement number.

2.9.2 BLC_PWM_CTL—Backlight PWM Control Register

BLC_PWM_CTL—Backlight PWM Control Register	
Register Type:	MMIO
Address Offset:	**48254h-48257h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

Bit	Description
31:16	**Reserved** Project: All Format:
15:0	**Backlight_Duty_Cycle** Project: All This field determines the number of time base events for the active portion of the PWM backlight control. This should never be larger than the frequency field. A value of zero will turn the backlight off. A value equal to the backlight modulation frequency field will be full on. This field gets updated when it is desired to change the intensity of the backlight, it will take affect at the end of the current PWM cycle. This value represents the active time of the PWM stream in PCH display raw clock periods multiplied by 128. This register must be written only as a full 32 bit dword. Byte or word writes are not supported.

2.9.3 BLM_HIST_CTL—Image Enhancement Histogram Control Register

BLM_HIST_CTL—Image Enhancement Histogram Control Register

Register Type: MMIO
Address Offset: **48260h-48263h**
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 32

Bit	Description
31	**Image_Enhancement_Histogram_Enabled** Project: All Default Value: 0b This bit enables the Image Enhancement histogram logic to collect data.

Value	Name	Description	Project
0b	Disable	Image histogram is disabled	All
1b	Enable	The image histogram is enabled. When this bit is changed from a zero to a one, histogram calculations will begin after the next VBLANK of the assigned pipe.	All

Bit	Description
30	**Image_Enhancement_Modification_Table_Enabled** Project: All Default Value 0b This bit enables the Image Enhancement modification table.

Value	Name	Description	Project
0b	Disable	Disabled	All
1b	Enable	Enabled. When this bit is changed from a zero to a one, modifications begin after the next VBLANK of the assigned pipe.	All

Bit	Description
29	**Image_Enhancement_Pipe_assignment** Project: All Default Value: 0b This bit assigns the IE function to a pipe. IE events will be synchronized to the VBLANK of the selected pipe. The IE function must be disabled in order to change the value of this field.

Value	Name	Description	Project
0b	Pipe A	Pipe A	All
1b	Pipe B	Pipe B	All

Bit	Description
28:25	**Reserved** Project: All Format:

BLM_HIST_CTL—Image Enhancement Histogram Control Register

24 — **Histogram Mode Select**

Project: All

Default Value: 0b

Value	Name	Description	Project
0b	YUV	YUV Luma Mode	All
1b	HSV	HSV Intensity Mode	All

23:16 — **Sync_to_Phase_In_Count** Project: All

This field indicates the phase in count number on which the Image Enhancement table will be loaded if the Sync to Phase in is enabled.

15 — **Reserved** Project: All Format:

14:13 — **Enhancement_mode**

Project: All

Default Value: 00b

Value	Name	Description	Project
00b	Direct	Direct look up mode	All
01b	Additive	Additive mode	All
10b	Multiplicative	Multiplicative mode	All
11b	Reserved	Reserved	All

12 — **Sync_to_Phase_In** Project: All

Setting this bit enables the double buffered registers to be loaded on the phase in count value specified instead of the next vblank.

11 — **Bin_Register_Function_Select**

Project: All

Default Value: 0b

This field indicates what data is being written to or read from the bin data register.

Value	Name	Description	Project
0b	BTC	Bin Threshold Count. A read from the bin data register returns that bin's threshold value from the most recent vblank load event (guardband threshold trip). Valid range for the Bin Index is 0 to 31.	All
1b	BIE	Bin Image Enhancement Value. Valid range for the Bin Index is 0 to 32	All

10:7 — **Reserved** Project: All Format:

6:0 — **Bin_Register_Index** Project: All

This field indicates the bin number whose data can be accessed through the bin data register. This value is automatically incremented by a read or a write to the bin data register if the busy bit is not set.

2.9.4 Image Enhancement Bin Data Register

Image Enhancement Bin Data Register(F0 Threshold Count Usage)	
Register Type:	MMIO
Address Offset:	**48264h-48267h**
Project:	All
Exists If:	BLM_HIST_CTL:Bin Register Function Select = 0
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32

Writes to this address are steered to the correct register by programming the Bin Register Function Select and the Bin Register Index.

Function 0 usage (Threshold Count) this Function is Read Only

Bit	Description
31	**Busy_Bit** Project: All If set , the engine is busy, the rest of the register is undefined. If clear, the register contains valid data.
30:22	**Reserved** Project: All Format:
21:0	**Bin_Count** Project: All The total number of pixels in this bin, value is updated at the start of each vblank.

Image Enhancement Bin Data Register(F1 Image Enhancement Usage)	
Register Type:	MMIO
Address Offset:	**48264h-48267h**
Project:	All
Exists If:	BLM_HIST_CTL:Bin Register Function Select = 1
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Next vblank if in normal mode, or on phase in Sync event frame if it is enabled

Writes to this address are steered to the correct register by programming the Bin Register Function Select and the Bin Register Index.

Function 1 usage (Image Enhancement) this Function is Read/Write

Bit	Description
31:10	**Reserved** Project: All Format:
9:0	**Image_Bin_Correction_Factor** Project: All The correction value for this bin. Writes to this register are double buffered on the next vblank if in normal mode, or on the phase in Sync event frame if it is enabled. The value written here is the 10bit corrected channel value for the lowest point of the bin.

2.9.5 Histogram Threshold Guardband Register

Histogram Threshold Guardband Register	
Register Type:	MMIO
Address Offset:	**48268h-4826Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank

Bit	Description
31	**Histogram_Interrupt_enable** Project: All Default Value: 0b

Value	Name	Description	Project
0b	Disable	Disabled	All
1b	Enable	This generates a histogram interrupt once a Histogram event occurs. Software must always program 1.	All

Bit	Description
30	**Histogram_Event_status** Project: All Access: R/W Clear Default Value: 0b When a Histogram event has occurred, this will get set by the hardware. For any more Histogram events to occur, the software needs to clear this bit by writing a '1'. The default state for this bit is '0'

Value	Name	Description	Project
0b	Not Occurred	Histogram event has not occurred	All
1b	Occurred	Histogram event has occurred	All

Bit	Description
29:22	**Guardband_Interrupt_Delay** Project: All An interrupt is always generated after this many consecutive frames of the guardband threshold being surpassed. This value is double buffered on start of vblank. A value of 0 is invalid.
21:0	**Threshold_Guardband** Project: All This value is used to determine the guardband for the threshold interrupt generation. This single value is used for all the segments. This value is double buffered on start of vblank

2.10 Motion Blur Mitigation (MBM) Control

These registers are use to control the MBM logic. (Sometimes called LOT, panel overdrive, or LRTC).

Before enabling MBM, software should have identically programmed both pipes source size and gamma tables. Additionally, before executing a flip on the selected pipe (which generates the MBM frame) software should have loaded the address of the reference frame into the second plane(s), this can be done by MMIO or by a flip command. The second pipe does not need to have it's panel fitter or FDI enabled (or anything else down the pipe from MBM).

2.10.1 MBM_CTRL—MBM Control

MBM_CTRL—MBM Control

Register Type: MMIO
Address Offset: **48800h-48803h**
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 32

Bit	Description
31	**MBM_Enable** Project: All Default Value: 0b This bit enables MBM logic.

Value	Name	Description	Project
0b	Disable	MBM is Disabled	All
1b	Enable	MBM is Enabled	All

Bit	Description
30	**Reserved** Project: All Format:
29	**MBM Pipe Select** Project: All Default Value: 0b This bit assigns MBM modification to the selected pipe.

Value	Name	Description	Project
0b	Pipe A	PipeA (PipeA will fetch the current buffer and PipeB will fetch the previous buffer)	All
1b	Pipe B	PipeB (PipeB will fetch the current buffer and PipeA will fetch the previous buffer)	All

MBM_CTRL—MBM Control	
28:27	**MBM_Surface_select** Project: All Default Value: 0b

Value	Name	Description	Project
00b	None	None	All
01b	Sprite	Sprite Only	All
10b	Primary	Primary Only	All
11b	Both	Both sprite and primary	All

26:24	**Reserved** Project: All Format:
23:16	**MBM_Delta_Threshold** Project: All Default Value: 00000000b If the delta value between the current and previous component values exceed this threshold a compensated value is generated. Otherwise, the current value is passed through.
15:0	**Reserved** Project: All Format:

2.10.2 MBM_TBL—MBM Overdrive Table

MBM_TBL—MBM Overdrive Table

Register Type: MMIO
Address Offset: **48810h-4890Bh**
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 63x32

These will be the overdrive values to be used as the lookup table entries for MBM. Correction table factors are stored x-major, groups of input values go together, ascending from row entry zero. The first row is internally hard coded to 0, so the first address actually corresponds to the second row of the table. The last row is internally hard coded to 256, so only 7 rows total are programmable.

DWord	Bit	Description
0..62	31:24	**Reserved** Project: All Format:
	23:16	**Red_MBM_overdrive_value** Project: All
	15:8	**Green_MBM_overdrive_value** Project: All
	7:0	**Blue_MBM_overdrive_value** Project: All

Overdrive table address offsets:

	0	31	63	95	127	159	191	223	255
0	0	0	0	0	0	0	0	0	0
31	48810h	48814h	48818h	4881Ch	48820h	48824h	48828h	4882Ch	48830h
63	48834h	48838h	4883Ch	48840h	48844h	48848h	4884Ch	48850h	48854h
95	48858h	4885Ch	48860h	48864h	48868h	4886Ch	48870h	48874h	48878h
127	4887Ch	48880h	48884h	48888h	4888Ch	48890h	48894h	48898h	4889Ch
159	488A0h	488A4h	488A8h	488ACh	488B0h	488B4h	488B8h	488BCh	488C0h
191	488C4h	488C8h	488CCh	488D0h	488D4h	488D8h	488DCh	488E0h	488E4h
223	488E8h	488ECh	488F0h	488F4h	488F8h	488FCh	48900h	48904h	48908h
255	256	256	256	256	256	256	256	256	256

Previous Pixel Value Range
Current Pixel Value Range
Address Offset
Hard Coded Value

2.11 Color Conversion and Control Registers

These registers contain the coefficients of the pipe color space converter. There are 12 values in 6 registers for each pipe. This color space conversion is used to convert the RGB frame buffer data into YUV data for use on the HDMI or DisplayPort. Or alternately a YUV frame buffer could be converted to RGB.

The high color channel is the most significant bits of the color. The low color channel is the least significant bits of the color. The medium color channel is the bits between high and low. For example: In RGB modes Red is in the High channel, Green in Medium, and Blue in Low. In YUV modes, U is in the High channel, Y in Medium, and V in Low.

The color space conversion registers are double buffered and are updated on the start of vertical blank following a write to the CSC Mode register for the respective pipe.

CSC Coefficient Description

Project: All
Default Value: 0000h
Size (in bits): 16

Coefficients for the CSC are stored in sign-exponent-mantissa format. Two CSC coefficients are stored in each dword, the table below show the data packing in each dword.

Bit	Description
15	**Sign** Project: All
14:12	**Exponent_bits** Project: All Represented as 2^{-n}
11:3	**Mantissa** Project: All
2:0	**Reserved** Project: All Format:

Sign

Value	Name	Description	Project
0b	Positive	Positive	All
1b	Negative	Negative	All

Exponent_bits

Value	Name	Description	Project
110b	4	4 or mantissa is bb.bbbbbbb	All
111b	2	2 or mantissa is b.bbbbbbbb	All
000b	1	1 or mantissa is 0.bbbbbbbbb	All
001b	0.5	0.5 or mantissa is 0.0bbbbbbbbb	All
010b	0.25	0.25 or mantissa is 0.00bbbbbbbbb	All
011b	0.125	0.125 or mantissa is 0.000bbbbbbbbb	All
others	Reserved	Reserved	All

The matrix equations are as follows:

Y = (RY * R) + (GY * G) + (BY * B)

U = (RU * R) + (GU * G) + (BU * B)

V = (RV * R) + (GV * G) + (BV * B)

The standard programming for RGB to YUV is in the following table.

	Bt.601		Bt.709	
	Value	Program	Value	Program
RU	0.2990	0x1990	0.21260	0x2D98
GU	0.5870	0x0968	0.71520	0x0B70
BU	0.1140	0x3E98	0.07220	0x3940
RV	-0.1687	0xAAC8	-0.11460	0xBEA8
GV	-0.3313	0x9A98	-0.38540	0x9C58
BV	0.5000	0x0800	0.50000	0x0800
RY	0.5000	0x0800	0.50000	0x0800
GY	-0.4187	0x9D68	-0.45420	0x9E88
BY	-0.0813	0xBA68	-0.04580	0xB5E0

The standard programming for YUV to sRGB without scaling is in the following table.

The input is VYU on high, medium, and low channels respectively.

The output is RGB on high, medium, and low channels respectively.

Program the pre-CSC offsets to -128, -16, and -128 for high, medium, and low channels respectively.

The coefficients can be scaled if desired.

	Bt.601 Reverse		Bt.709 Reverse	
	Value	Program	Value	Program
GY	1.000	0x7800	1.000	0x7800
BY	0.000	0x0000	0.000	0x0000
RY	1.371	0x7AF8	1.574	0x7C98
GU	1.000	0x7800	1.000	0x7800
BU	-0.336	0x9AC0	-0.187	0xABF8
RU	-0.698	0x8B28	-0.468	0x9EF8
GV	1.000	0x7800	1.000	0x7800
BV	1.732	0x7DD8	1.855	0x7ED8
RV	0.000	0x0000	0.000	0x0000

2.11.1 Pipe A Color Control

2.11.1.1 CSC_A_Coefficients 1

CSC_A_Coefficients 1	
Register Type:	MMIO
Address Offset:	**49010h-49013h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to CSC_A_Mode

Bit	Description
31:16	**RY** Project: All Format: CSC COEFFICIENT DESCRIPTION FormatDesc: Coefficients for the CSC are stored in sign-exponent-mantissa format. Two CSC coefficients are stored in each dword, the table below show the data packing in each dword. CSC coefficient. See format description above.
15:0	**GY** Project: All Format: CSC COEFFICIENT DESCRIPTION FormatDesc: Coefficients for the CSC are stored in sign-exponent-mantissa format. Two CSC coefficients are stored in each dword, the table below show the data packing in each dword. CSC coefficient. See format description above.

2.11.1.2 CSC_A_Coefficients 2

CSC_A_Coefficients 2	
Register Type:	MMIO
Address Offset:	**49014h-49017h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to CSC_A_Mode

Bit	Description
31:16	**BY** Project: All Format: CSC COEFFICIENT DESCRIPTION FormatDesc: Coefficients for the CSC are stored in sign-exponent-mantissa format. Two CSC coefficients are stored in each dword, the table below show the data packing in each dword. CSC coefficient. See format description above.
15:0	**Reserved** Project: All Format:

2.11.1.3 CSC_A_Coefficients 3

CSC_A_Coefficients 3	
Register Type:	MMIO
Address Offset:	**49018h-4901Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to CSC_A_Mode

Bit	Description
31:16	**RU** Project: All Format: CSC COEFFICIENT DESCRIPTION FormatDesc: Coefficients for the CSC are stored in sign-exponent-mantissa format. Two CSC coefficients are stored in each dword, the table below show the data packing in each dword. CSC coefficient. See format description above.
15:0	**GU** Project: All Format: CSC COEFFICIENT DESCRIPTION FormatDesc: Coefficients for the CSC are stored in sign-exponent-mantissa format. Two CSC coefficients are stored in each dword, the table below show the data packing in each dword. CSC coefficient. See format description above.

2.11.1.4 CSC_A_Coefficients 4

CSC_A_Coefficients 4	
Register Type:	MMIO
Address Offset:	**4901Ch-4901Fh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to CSC_A_Mode

Bit	Description
31:16	**BU** Project: All Format: CSC COEFFICIENT DESCRIPTION FormatDesc: Coefficients for the CSC are stored in sign-exponent-mantissa format. Two CSC coefficients are stored in each dword, the table below show the data packing in each dword. CSC coefficient. See format description above.
15:0	**Reserved** Project: All Format:

2.11.1.5 CSC_A_Coefficients 5

CSC_A_Coefficients 5	
Register Type:	MMIO
Address Offset:	**49020h-49023h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to CSC_A_Mode

Bit	Description
31:16	**RV** Project: All Format: CSC COEFFICIENT DESCRIPTION FormatDesc: Coefficients for the CSC are stored in sign-exponent-mantissa format. Two CSC coefficients are stored in each dword, the table below show the data packing in each dword. CSC coefficient. See format description above.
15:0	**GV** Project: All Format: CSC COEFFICIENT DESCRIPTION FormatDesc: Coefficients for the CSC are stored in sign-exponent-mantissa format. Two CSC coefficients are stored in each dword, the table below show the data packing in each dword. CSC coefficient. See format description above.

2.11.1.6 CSC_A_Coefficients 6

CSC_A_Coefficients 6	
Register Type:	MMIO
Address Offset:	**49024h-49027h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to CSC_A_Mode

Bit	Description
31:16	**BV** Project: All Format: CSC COEFFICIENT DESCRIPTION FormatDesc: Coefficients for the CSC are stored in sign-exponent-mantissa format. Two CSC coefficients are stored in each dword, the table below show the data packing in each dword. CSC coefficient. See format description above.
15:0	**Reserved** Project: All Format:

2.11.1.7 CSC_A_Mode

CSC_A_Mode

Register Type:	MMIO
Address Offset:	**49028h-4902Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank

Writes to this register arm CSC_A registers

Bit	Description
31:3	**Reserved** Project: All Format:
2	**CSC_Black_Screen_Offset** Project: All Default Value: 0b Adds an offset to the data output from CSC In sRGB ouput mode: RGB is defined as R`+ 1/16, G`+ 1/16, B`+ 1/16 In rcYUV output mode: YUV is defined as Y'+ 1/16, U and V are output in excess 2048 format
1	**CSC_Position** Project: All Default Value: 0b Selects the CSC position in the pipe.
0	**CSC_Mode** Project: All Default Value: 0b Selects the CSC direction.

Value	Name	Description	Project
0b	No Offset	CSC output has no offset added (will be RGB or YUV, depending on bit 0)	All
1b	Offset	CSC output has offset added (will be sRGB or rcYUV depending on bit 0)	All

Value	Name	Description	Project
0b	CSC After	CSC is after gamma and DPST image enhancement	All
1b	CSC Before	CSC is before gamma and DPST image enhancement	All

Value	Name	Description	Project
0b	RGB to YUV	RGB to YUV conversion	All
1b	YUV to RGB	YUV to RGB conversion	All

2.11.1.8 Pre-CSC_A High Color Channel Offset

Pre-CSC_A High Color Channel Offset	
Register Type:	MMIO
Address Offset:	**49030h-49033h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to CSC_A_Mode

Bit	Description
31:13	**Reserved** Project: All Format:
12:0	**Pre-CSC_High_Color_Channel_Offset** Project: All This 13-bit 2's complement value is used to give an offset to the color channel as it enters CSC logic.

2.11.1.9 Pre-CSC_A Medium Color Channel Offset

Pre-CSC_A Medium Color Channel Offset	
Register Type:	MMIO
Address Offset:	**49034h-49037h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to CSC_A_Mode

Bit	Description
31:13	**Reserved** Project: All Format:
12:0	**Pre-CSC_Medium_Color_Channel_Offset** Project: All This 13-bit 2's complement value is used to give an offset to the color channel as it enters CSC logic.

2.11.1.10 Pre-CSC_A Low Color Channel Offset

Pre-CSC_A Low Color Channel Offset	
Register Type:	MMIO
Address Offset:	**49038h-4903Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to CSC_A_Mode

Bit	Description
31:13	**Reserved** Project: All Format:
12:0	**Pre-CSC_Low_Color_Channel_Offset** Project: All This 13-bit 2's complement value is used to give an offset to the color channel as it enters CSC logic.

2.11.2 Pipe B Color Control

2.11.2.1 CSC_B_Coefficients 1

CSC_B_Coefficients 1	
Register Type:	MMIO
Address Offset:	**49110h-49113h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to CSC_B_Mode

Bit	Description
31:16	**RY** Project: All Format: CSC COEFFICIENT DESCRIPTION FormatDesc: Coefficients for the CSC are stored in sign-exponent-mantissa format. Two CSC coefficients are stored in each dword, the table below show the data packing in each dword. CSC coefficient. See format description above.

CSC_B_Coefficients 1	
15:0	**GY** Project: All Format: CSC COEFFICIENT DESCRIPTION FormatDesc: Coefficients for the CSC are stored in sign-exponent-mantissa format. Two CSC coefficients are stored in each dword, the table below show the data packing in each dword. CSC coefficient. See format description above.

2.11.2.2 CSC_B_Coefficients 2

CSC_B_Coefficients 2	
Register Type:	MMIO
Address Offset:	**49114h-49117h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to CSC_B_Mode

Bit	Description
31:16	**BY** Project: All Format: CSC COEFFICIENT DESCRIPTION FormatDesc: Coefficients for the CSC are stored in sign-exponent-mantissa format. Two CSC coefficients are stored in each dword, the table below show the data packing in each dword. CSC coefficient. See format description above.
15:0	**Reserved** Project: All Format:

2.11.2.3 CSC_B_Coefficients 3

CSC_B_Coefficients 3	
Register Type:	MMIO
Address Offset:	**49118h-4911Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to CSC_B_Mode

Bit	Description
31:16	**RU** Project: All Format: CSC COEFFICIENT DESCRIPTION FormatDesc: Coefficients for the CSC are stored in sign-exponent-mantissa format. Two CSC coefficients are stored in each dword, the table below show the data packing in each dword. CSC coefficient. See format description above.
15:0	**GU** Project: All Format: CSC COEFFICIENT DESCRIPTION FormatDesc: Coefficients for the CSC are stored in sign-exponent-mantissa format. Two CSC coefficients are stored in each dword, the table below show the data packing in each dword. CSC coefficient. See format description above.

2.11.2.4 CSC_B_Coefficients 4

CSC_B_Coefficients 4	
Register Type:	MMIO
Address Offset:	**4911Ch-4911Fh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to CSC_B_Mode

Bit	Description
31:16	**BU** Project: All Format: CSC COEFFICIENT DESCRIPTION FormatDesc: Coefficients for the CSC are stored in sign-exponent-mantissa format. Two CSC coefficients are stored in each dword, the table below show the data packing in each dword. CSC coefficient. See format description above.
15:0	**Reserved** Project: All Format:

2.11.2.5 CSC_B_Coefficients 5

CSC_B_Coefficients 5	
Register Type:	MMIO
Address Offset:	**49120h-49123h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to CSC_B_Mode

Bit	Description
31:16	**RV** Project: All Format: CSC COEFFICIENT DESCRIPTION FormatDesc: Coefficients for the CSC are stored in sign-exponent-mantissa format. Two CSC coefficients are stored in each dword, the table below show the data packing in each dword. CSC coefficient. See format description above.
15:0	**GV** Project: All Format: CSC COEFFICIENT DESCRIPTION FormatDesc: Coefficients for the CSC are stored in sign-exponent-mantissa format. Two CSC coefficients are stored in each dword, the table below show the data packing in each dword. CSC coefficient. See format description above.

2.11.2.6 CSC_B_Coefficients 6

CSC_B_Coefficients 6	
Register Type:	MMIO
Address Offset:	**49124h-49127h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to CSC_B_Mode

Bit	Description
31:16	**BV** Project: All Format: CSC COEFFICIENT DESCRIPTION FormatDesc: Coefficients for the CSC are stored in sign-exponent-mantissa format. Two CSC coefficients are stored in each dword, the table below show the data packing in each dword. CSC coefficient. See format description above.
15:0	**Reserved** Project: All Format:

2.11.2.7 CSC_B_Mode

CSC_B_Mode	
Register Type:	MMIO
Address Offset:	**49128h-4912Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank

Writes to this register arm CSC_B registers

Bit	Description
31:3	**Reserved** Project: All Format:

CSC_B_Mode

2

CSC_Black_Screen_Offset

Project: All

Default Value: 0b

Adds an offset to the data output from CSC

In sRGB ouput mode: RGB is defined as R`+ 1/16, G`+ 1/16, B`+ 1/16

In rcYUV output mode: YUV is defined as Y'+ 1/16, U and V are output in excess 2048 format

Value	Name	Description	Project
0b	No Offset	CSC output has no offset added (will be RGB or YUV, depending on bit 0)	All
1b	Offset	CSC output has offset added (will be sRGB or rcYUV depending on bit 0)	All

1

CSC_Position

Project: All

Default Value: 0b

Selects the CSC position in the pipe.

Value	Name	Description	Project
0b	CSC After	CSC is after gamma and DPST image enhancement	All
1b	CSC Before	CSC is before gamma and DPST image enhancement	All

0

CSC_Mode

Project: All

Default Value: 0b

Selects the CSC direction. Input and output formats and position within the pipe

Value	Name	Description	Project
0b	RGB to YUV	RGB to YUV conversion	All
1b	YUV to RGB	YUV to RGB conversion	All

2.11.2.8 Pre-CSC_B High Color Channel Offset

Pre-CSC_B High Color Channel Offset			
Register Type:	MMIO		
Address Offset:	**49130h-49133h**		
Project:	All		
Default Value:	00000000h		
Access:	R/W		
Size (in bits):	32		
Double Buffer Update Point:	Start of vertical blank after armed		
Double Buffer Armed By:	Write to CSC_B_Mode		
Bit	**Description**		
31:13	**Reserved** Project: All	Format:	
12:0	**Pre-CSC_B_High_Color_Channel_Offset**	Project: All	
	This 13-bit 2's complement value is used to give an offset to the color channel as it enters CSC logic.		

2.11.2.9 Pre-CSC_B Medium Color Channel Offset

Pre-CSC_B Medium Color Channel Offset			
Register Type:	MMIO		
Address Offset:	**49134h-49137h**		
Project:	All		
Default Value:	00000000h		
Access:	R/W		
Size (in bits):	32		
Double Buffer Update Point:	Start of vertical blank after armed		
Double Buffer Armed By:	Write to CSC_B_Mode		
Bit	**Description**		
31:13	**Reserved** Project: All	Format:	
12:0	**Pre-CSC_B_Medium_Color_Channel_Offset**	Project: All	
	This 13-bit 2's complement value is used to give an offset to the color channel as it enters CSC logic.		

2.11.2.10 Pre-CSC_B Low Color Channel Offset

Pre-CSC_B Low Color Channel Offset			
Register Type:	MMIO		
Address Offset:	**49138h-4913Bh**		
Project:	All		
Default Value:	00000000h		
Access:	R/W		
Size (in bits):	32		
Double Buffer Update Point:	Start of vertical blank after armed		
Double Buffer Armed By:	Write to CSC_B_Mode		
Bit	**Description**		
31:13	**Reserved** Project: All	Format:	
12:0	**Pre-CSC_B_Low_Color_Channel_Offset** Project: All This 13-bit 2's complement value is used to give an offset to the color channel as it enters CSC logic.		

2.12 Display Palette Registers (4A000h–4CFFFh)

The display palette provides a means to correct the gamma of an image stored in a frame buffer to match the gamma of the monitor or presentation device. Additionally, the display palette provide a method for converting indexed data values to color values for VGA and 8-bpp indexed display modes. The display palette is located after the plane blender. Using the individual plane gamma enables, the blended pixels can go through or bypass the palette on a pixel by pixel basis.

The display palette can be accessed through multiple methods and operate in one of three different modes.

8 bit legacy palette mode (for indexed VGA and 8 bpp formats and for legacy gamma):

256 entries of 24 bits each (8 bits per color).

For indexed formats, an 8 bit per pixel value is used to lookup a 24 bit per pixel value from the palette which then is padded to 36 bits. This permits a compact data format to choose from 256 colors out of a larger palette of colors. The legacy palette is accessible through both MMIO and VGA palette register I/O addresses. Through VGA palette register I/O addresses, the palette can look as though there are only 6 bits per color component (this mapping is handled inside the VGA engine).

For legacy gamma, the 36 bits per pixel gamma input is chopped to 24 bits and used to lookup a 24 bit pixel value from the palette which then is padded to 36 bits. This permits a color to be re-mapped to a different brightness for gamma correction. This provides the lowest quality gamma and should only be used for legacy requirements.

10 bit precision palette mode (for 10 bit gamma):

1024 entries of 30 bits each (10 bits per color).

For 10 bit gamma, the 36 bits per pixel gamma input is chopped to 30 bits and used to lookup a 30 bit pixel value from the palette which then is padded to 36 bits. This permits a color to be re-mapped to a different brightness for gamma correction. This provides the highest quality gamma for non-indexed pixel data formats of 30 bits per pixel or less.

12 bit interpolated gamma mode:

512 entries of 16 bits each (format described in 12 bit interpolated gamma programming notes).

For 12 bit interpolated gamma, the 36 bits per pixel gamma input is used to lookup reference points (16 bits per color in 12.4 format) along a gamma curve and interpolate a 36 bit pixel result. This permits a color to be re-mapped to a different brightness for gamma correction. This provides the highest quality gamma for pixel data formats greater than 30 bits per pixel.

Pixel chopping refers to removing the LSBs of each color component to reduce bits per pixel. Pixel padding refers to adding LSBs to each color component to increase bits per pixel.

Accesses to the palette entries require that the core display clock is running at the time of the update. Do not access the palette if the pipe is enabled and mode set is not yet complete. All write accesses to the palette must be in dwords. Byte or word writes to the palettes are not allowed.

2.12.1 LGC_PALETTE_A—Pipe A Legacy Display Palette

LGC_PALETTE_A—Pipe A Legacy Display Palette					
Register Type:	MMIO				
Address Offset:	**4A000h-4A3FFh**				
Project:	All				
Default Value:	UUUUUUUUh				
Access:	R/W (DWORD access only, no byte access)				
Size (in bits):	256x32				
DWord	**Bit**	**Description**			
0..255	31:24	**Reserved**	Project: All		Format:
	23:16	**Red_Palette_Entry**	Project: All		Format:
	15:8	**Green_Palette_Entry**	Project: All		Format:
	7:0	**Blue_Palette_Entry**	Project: All		Format:

2.12.2 LGC_PALETTE_B—Pipe B Legacy Display Palette

LGC_PALETTE_B—Pipe B Legacy Display Palette					
Register Type:	MMIO				
Address Offset:	**4A800h-4ABFFh**				
Project:	All				
Default Value:	UUUUUUUUh				
Access:	R/W (DWORD access only, no byte access)				
Size (in bits):	256x32				
DWord	**Bit**	**Description**			
0..255	31:24	**Reserved**	Project: All		Format:
	23:16	**Red_Palette_Entry**	Project: All	Format:	
	15:8	**Green_Palette_Entry**	Project: All	Format:	
	7:0	**Blue_Palette_Entry**	Project: All	Format:	

2.12.3 PREC_PALETTE_A—Pipe A Precision Display Palette

10 bit Precision Palette Mode Format				
Project:	All			
Format for 10 bit precision palette mode.				
Bit	**Description**			
31:30	**Reserved**	Project: All	Format:	
29:20	**Red_Palette_Entry**		Project:	All
19:10	**Green_Palette_Entry**		Project:	All
9:0	**Blue_Palette_Entry**		Project:	All

12-bit Interpolated Precision Palette Mode (odd Dword) Format				
Project:	All			
Format for 12 bit interpolated gamma mode, odd dwords.				
Bit	**Description**			
31:30	**Reserved**	Project: All	Format:	
29:20	**Red_ Base[11:2]**		Project:	All
19:10	**Green_ Base[11:2]**		Project:	All
9:0	**Blue_ Base[11:2]**		Project:	All

12-bit Interpolated Precision Palette Mode (even Dword) Format

Project: All

Format for 12 bit interpolated gamma mode, odd dwords..

Bit	Description					
31:30	**Reserved**	Project:	All		Format:	MBZ
29:28	**Red_Base[1:0]**				Project:	All
27:24	**Red_Fraction**				Project:	All
23:20	**Reserved**	Project:	All		Format:	
19:18	**Green_Base[1:0]**				Project:	All
17:14	**Green_Fraction**				Project:	All
13:10	**Reserved**	Project:	All		Format:	
9:8	**Blue_Base[1:0]**				Project:	All
7:4	**Blue_Fraction**				Project:	All
3:0	**Reserved**	Project:	All		Format:	

PREC_PALETTE_A—Pipe A Precision Display Palette(10 bit)

Register Type: MMIO
Address Offset: **4B000h-4BFFFh**
Project: All
Exists If: PIPEACONF:Pipe_A_Palette/Gamma_Unit_mode = 01b
Default Value: UUUUUUUUh
Access: R/W (DWORD access only, no byte access)
Size (in bits): 1024x32

DWord	Bit	Description				
0..1023	31:0	**10bit_mode** See format description above	Project:	All	Format:	10 bit Precision Palette Mode Format

PREC_PALETTE_A—Pipe A Precision Display Palette(12 bit)		
Register Type:	MMIO	
Address Offset:	**4B000h-4BFFFh**	
Project:	All	
Exists If:	PIPEACONF:Pipe_A_Palette/Gamma_Unit_mode = 10b	
Default Value:	UUUUUUUUh	
Access:	R/W (DWORD access only, no byte access)	
Size (in bits):	1024x32	
DWord	**Bit**	**Description**
0..1023	63:32	**12bit_odd** Project: All Format: 12-bit Interpolated Precision Palette Mode (odd Dword) Format FormatDesc: Format for 12 bit interpolated gamma mode, odd dwords.
	31:0	**12bit_even** Project: All Format: 12-bit Interpolated Precision Palette Mode (even Dword) Format FormatDesc: Format for 12 bit interpolated gamma mode dwords

2.12.4 PREC_PALETTE_B—Pipe B Precision Display Palette

PREC_PALETTE_B—Pipe B Precision Display Palette(10 bit)		
Register Type:	MMIO	
Address Offset:	**4C000h-4CFFFh**	
Project:	All	
Exists If:	PIPEBCONF:Pipe_B_Palette/Gamma_Unit_mode = 01b	
Default Value:	UUUUUUUUh	
Access:	R/W (DWORD access only, no byte access)	
Size (in bits):	1024x32	
DWord	**Bit**	**Description**
0..1023	31:0	**10bit_mode** Project: All Format: 10 bit Precision Palette Mode Format FormatDesc: Format for 10 bit precision palette mode.

PREC_PALETTE_B—Pipe B Precision Display Palette(12 bit)

Register Type: MMIO
Address Offset: **4C000h-4CFFFh**
Project: All
Exists If: PIPEBCONF:Pipe_B_Palette/Gamma_Unit_mode = 10b
Default Value: UUUUUUUUh
Access: R/W (DWORD access only, no byte access)
Size (in bits): 1024x32

DWord	Bit	Description
0..1023	63:32	**12bit_odd** Project: All Format: 12-bit Interpolated Precision Palette Mode (odd Dword) Format Format Desc: Format for 12 bit interpolated gamma mode, odd dwords.
	31:0	**12bit_even** Project: All Format: 12-bit Interpolated Precision Palette Mode (even Dword) Format FormatDesc: Format for 12 bit interpolated gamma mode dwords.

12-bit Interpolated Gamma Programming Notes:

The 12-bit gamma correction curve is represented by specifying a set of reference points spaced equally along the desired curve. Red, Green, and Blue each have 513 reference points. The first 512 reference points are stored in the precision palette RAM, and the final value is stored in the GCMAX register. The first 512 reference points are 16 bits represented in a 12.4 format with 12 integer and 4 fractional bits. The final reference points are 17 bits represented in a 13.4 format with 13 integer and 4 fractional bits.

During operation the appropriate reference point pairs (adjacent) are selected for each color, and the output is interpolated between these two reference point values.

To program the gamma correction reference points, calculate the desired gamma curve for inputs from 0 to 4096. Every 8th point on the curve (0, 8, 16 ... 4088, 4096) becomes a reference point. Convert the gamma value to the 13.4 format. The first 512 reference points are saved to the precision palette RAM, where the even DWords contain the lower 6 bits of the reference point value, and the odd DWords contain the upper 10 bits of the reference point value. The final 513th reference point is saved in the GCMAX registers in 13.4 format.

Example equation for gamma curve of 2.2:

For (X = 0..4096) { gamma = [(X / 4096) ^ 2.2] * 4096 }

The curve must be flat or increasing, never decreasing.

2.12.5 PIPEAGCMAX—Pipe A Gamma Correction Max

Pipe Max Gamma Correction Format	
Project: All **Default Value:** 00010000h	
Bit	**Description**
31:17	**Reserved** Project: All Format:
16:0	**Max_Color_Gamma_Correction_Point** Project: All 513th reference point for the color channel of the 12-bit pipe piecewise linear gamma correction. The value should always be programmed to be less than or equal to 4096.0.

PIPEAGCMAX—Pipe A Gamma Correction Max						
Register Type: MMIO **Address Offset:** **4D000h-4D00Bh** **Project:** All **Default Value:** 00010000h **Access:** R/W **Size (in bits):** 3x32						
DWord	**Bit**	**Description**				
0	31:0	**Red**	Project:	All	Format:	Pipe Max Gamma Correction Format
1	31:0	**Green**	Project:	All	Format:	Pipe Max Gamma Correction Format
2	31:0	**Blue**	Project:	All	Format:	Pipe Max Gamma Correction Format

2.12.6 PIPEBGCMAX —Pipe B Gamma Correction Max

PIPEBGCMAX—Pipe B Gamma Correction Max						
Register Type: MMIO **Address Offset:** **4D010h-4D01Bh** **Project:** All **Default Value:** 00010000h **Access:** R/W **Size (in bits):** 3x32						
DWord	**Bit**	**Description**				
0	31:0	**Red**	Project:	All	Format:	Pipe Max Gamma Correction Format
1	31:0	**Green**	Project:	All	Format:	Pipe Max Gamma Correction Format
2	31:0	**Blue**	Project:	All	Format:	Pipe Max Gamma Correction Format

2.13 Software Flag Registers (4F000h–4F10Fh)

2.13.1 Software Flag Registers

Software Flag Registers						
Register Type: MMIO **Address Offset:** **4F000h-4F08Fh** **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 36x32						
These registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these registers is defined by the software architecture.						
DWord	**Bit**	**Description**				
0..35	31:0	**Reserved**	Project:	All	Format:	PBC

2.13.2 DE_LOAD_SL — Display Load Scan Lines

DE_LOAD_SL—Display Load Scan Lines	
Register Type:	MMIO
Address Offset:	**4F100h-4F103h**
Project:	**DevSNB**
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

This register is used to initiate a display scan line compare on DevSNB:D2 and later steppings.

When this register is written with the Init_Compare bit set to 1b, the Display Engine (DE) will start comparing the display pipe timing generator current scan line value (current scan line) with the start scan line value (current scan line >= start scan line) and the end scan line value (current scan line <= end scan line) to decide if the the pipe scan line is inside or outside the scan line window of interest. DE will wait until the current scan line is either outside (Inclusive mode) or inside (Exclusive mode) the scan line window, then trigger a scan line event and stop any further comparing.

The scan line event can cause display to send a scan line compare response to the render command streamer, used for releasing a MI_WAIT_FOR_EVENT on scan line window, if unmasked in the DERRMR mask register 0x44050. The scan line event can also cause display to generate a scan line compare interrupt, if unmasked and enabled in the DEIMR and DEIER display engine interrupt registers.

The end scan line value must be greater than or equal to the start scan line value.

The current scan line is a 13 bit value, but the scan line comparison is only done on the upper 10 bits. This gives an 8 line granularity on the compare.

The programmable range can include the vertical blank.

In interlaced display timings, the current scan line is the the current line of the current interlaced field.

Notes for command streamer programming to use this display load scan lines register:

- Either MMIO or a MI_LOAD_REGISTER_IMM command can be used to unmask the scan line render response 0x44050. That can be done any time before programming this register.
- In order to use MI_WAIT_FOR_EVENT on scan line window, DE_LOAD_SL must be programmed using MI_LOAD_REGISTER_IMM immediately prior to the MI_WAIT_FOR_EVENT on scan line window, both commands must be in the same cacheline, both commands must be executed using the same tail or batch update, and if sync flush is enabled, MI_SUSPEND_FLUSH must be used to suspend flushes prior to the commands.

DE_LOAD_SL—Display Load Scan Lines

Bit	Description
31	**Initiate_Compare** Default Value: 0b This field initiates the scan line compare. When this register is written with this bit set to 1b, the display engine will do one complete comparison cycle, trigger a scan line event, then stop comparing. Do not write this register again until after any previous scan line compare has completed. A compare can not be cancelled by writing with this bit set to 0b.

Value	Name	Description	Project
0b	Do nothing	Do nothing	All
1b	Initiate	Initiate the scan line compare.	All

Bit	Description
30	**Inclusive_Exclusive_Select** Default Value: 0b This field selects whether the scan line compare is done in exclusive mode, where display triggers the scan line event when inside the window (wait while outside window), or inclusive mode, where display triggers the scan line event when outside the scan line window (wait while inside window).

Value	Name	Description	Project
0b	Exclusive	Exclusive mode: trigger scan line event when inside the scan line window (wait while outside window)	All
1b	Inclusive	Inclusive mode: trigger scan line event when outside the scan line window (wait while inside window)	All

Bit	Description
29	**Pipe_Select** Default Value: 0b This field selects which display pipe timing generator scan line value to use for the compare.

Value	Name	Description	Project
0b	Pipe A	Display Pipe A	All
1b	Pipe B	Display Pipe B	All

Bit	Description
28:19	**Start_Scan_Line** Project: DevSNB Format: This field specifies the upper 10 bits of the starting scan line number of the scan line window. The lower 3 bits are reserved below and not programmable and not compared.
18:13	**Reserved** Project: All Format:
12:3	**End_Scan_Line** Project: DevSNB Format: This field specifies the upper 10 bits of the ending scan line number of the scan line window. The lower 3 bits are reserved below and not programmable and not compared.
2:0	**Reserved** Project: All Format:

3. North Pipe and Port Controls (60000h–6FFFFh)

3.1 Pipe A Timing

3.1.1 HTOTAL_A—Pipe A Horizontal Total Register

HTOTAL_A—Pipe A Horizontal Total Register	
Register Type:	MMIO
Address Offset:	**60000h-60003h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

Bit	Description
31:29	**Reserved** Project: All Format: MBZ
28:16	**Pipe_A_Horizontal_Total_Display_Clocks** Project: All This 13-bit field provides Horizontal Total up to 8192 pixels encompassing the Horizontal Active Display period, front/back border and retrace period. This field is programmed to the number of clocks desired minus one. This number of clocks needs to be a multiple of two when driving the LVDS port in two channel mode. This value should always be equal or greater to the sum of the horizontal active and the horizontal blank, and border region sizes.
15:12	**Reserved** Project: All Format: MBZ
11:0	**Pipe_A_Horizontal_Active_Display_Pixels** Project: All This 12-bit field provides Horizontal Active Display resolutions up to 4096 pixels. Note that the first horizontal active display pixel is considered pixel number 0. The value programmed should be the (active pixels/line – 1). The number of active pixels will be limited to multiples of two pixels when driving the integrated LVDS port in two channel mode. The minimum horizontal active display size allowed will be 64 pixels.

3.1.2 HBLANK_A—Pipe A Horizontal Blank Register

HBLANK_A—Pipe A Horizontal Blank Register	
Register Type:	MMIO
Address Offset:	**60004h-60007h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

Bit	Description
31:29	**Reserved** Project: All Format:
28:16	**Pipe_A_Horizontal_Blank_End** Project: All This 13-bit field specifies the position of Horizontal Blank End expressed in terms of the absolute pixel number relative to the horizontal active display start. The value programmed should be the HBLANK End pixel position, where the first active pixel is considered position 0; the second active pixel is considered position 1, etc. Horizontal blank ending at the same point as the horizontal total indicates that there is no left hand border area. HBLANK size has a minimum value of 32 clocks. The number of clocks within blank needs to be a multiple of two when driving the LVDS port in two channel mode. The value loaded in the register would be equal to RightBorder+Active+HBlank-1. The border must be 0, so this register must always be programmed to the same value as the Horizontal Total.
15:13	**Reserved** Project: All Format:
12:0	**Pipe_A_Horizontal_Blank_Start** Project: All This 13-bit field specifies the Horizontal Blank Start position expressed in terms of the absolute pixel number relative to the horizontal active display start. The value programmed should be the HBLANK Start pixel position, where the first active pixel is considered position 0; the second active pixel is considered position 1, etc. The number of clocks for both left and right borders need to be a multiple of two when driving the LVDS port in two channel mode. Horizontal blank should only start after the end of the horizontal active region. The value loaded in the register would be equal to RightBorder+Active-1. The border must be 0, so this register must always be programmed to the same value as the Horizontal Active.

3.1.3 HSYNC_A—Pipe A Horizontal Sync Register

HSYNC_A—Pipe A Horizontal Sync Register	
Register Type:	MMIO
Address Offset:	**60008h-6000Bh**
Project:	All
Default Value:	00000000h
Size (in bits):	32
Bit	**Description**
31:29	**Reserved** Project: All Format: MBZ
28:16	**Pipe_A_Horizontal_Sync_End** Project: All Default Value: 0b This 13-bit field specifies the horizontal Sync End position expressed in terms of the absolute pixel number relative to the horizontal active display start. The value programmed should be the HSYNC End pixel position, where the first active pixel is considered position 0; the second active pixel is considered position 1, etc. The number of clocks in the sync period needs to be a multiple of two when driving the LVDS port in two channel mode. This value should be greater than the horizontal sync start position and would be loaded with the Active+RightBorder+FrontPorch+Sync-1.
15:13	**Reserved** Project: All Format: MBZ
12:0	**Pipe_A_Horizontal_Sync_Start** Project: All Default Value: 0b This 13-bit field specifies the horizontal Sync Start position expressed in terms of the absolute pixel number relative to the horizontal active display start. The value programmed should be the HSYNC Start pixel position, where the first active pixel is considered position 0; the second active pixel is considered position 1, etc. Note that when HSYNC Start is programmed equal to HBLANK Start, both HSYNC and HBLANK will be asserted on the same pixel clock. It should never be programmed to less than HBLANK start. The number of cycles from the beginning of the line needs to be a multiple of two when driving the LVDS port in two channel mode. This register should not be less than the horizontal active end. This register should be loaded with the Active+RightBorder+FrontPorch-1.

3.1.4 VTOTAL_A—Pipe A Vertical Total Register

VTOTAL_A—Pipe A Vertical Total Register	
Register Type:	MMIO
Address Offset:	**6000Ch-6000Fh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

Bit	Description
31:29	**Reserved** Project: All Format:
28:16	**Pipe_A_Vertical_Total_Display_Lines** Project: All This 13 bit field provides Vertical Total up to 8192 lines encompassing the Vertical Active Display Lines, top/bottom border and retrace period. The value programmed should be the number of lines required minus one. Vertical total needs to be large enough to be greater than the sum of the vertical active, vertical border, and the vertical blank regions. The vertical counter is incremented on the leading edge of the horizontal sync. For interlaced display modes, this indicates the total number of lines in both fields. In interlaced modes, hardware automatically divides this number by 2 to get the number of lines in each field.
15:12	**Reserved** Project: All Format:
11:0	**Pipe_A_Vertical_Active_Display_Lines** Project: All This 12-bit field provides vertical active display resolutions up to 4096 lines. It should be programmed with the desired number of lines minus one. When using the internal panel fitting logic, the minimum vertical active area must be seven lines. For interlaced display modes, this indicates the total number of lines in both fields. In interlaced modes, hardware automatically divides this number by 2 to get the number of lines in each field.

3.1.5 VBLANK_A—Pipe A Vertical Blank Register

VBLANK_A—Pipe A Vertical Blank Register	
Register Type:	MMIO
Address Offset:	**60010h-60013h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

Bit	Description
31:29	**Reserved** Project: All Format:
28:16	**Pipe_A_Vertical_Blank_End** Project: All This 13-bit field specifies the Vertical Blank End position expressed in terms of the absolute Line number relative to the vertical active display start. The value programmed should be the VBLANK End line position, where the first active line is considered line 0, the second active line is considered line 1, etc. The end of vertical blank should be after the start of vertical blank and before or equal to the vertical total. This register should be loaded with the Vactive+BottomBorder+VBlank-1. For interlaced display modes, hardware automatically divides this number by 2 to get the vertical blank end in each field. It does not count the two half lines that get added when operating in modes with half lines. The border must be 0, so this register must always be programmed to the same value as the Vertical Total.
15:13	**Reserved** Project: All Format:
12:0	**Pipe_A_Vertical_Blank_Start** Project: All This 13-bit field specifies the Vertical Blank Start expressed in terms of the absolute line number relative to the vertical active display start. The value programmed should be the VBLANK Start line position, where the first active line is considered line 0, the second active line is considered line 1, etc. Minimum vertical blank size is required to be at least three lines. Blank should start after the end of active. This register is loaded with the Vactive+BottomBorder-1. For interlaced display modes, hardware automatically divides this number by 2 to get the vertical blank start in each field. It does not count the two half lines that get added when operating in modes with half lines. The border must be 0, so this register must always be programmed to the same value as the Vertical Active.

3.1.6 VSYNC_A—Pipe A Vertical Sync Register

VSYNC_A—Pipe A Vertical Sync Register	
Register Type:	MMIO
Address Offset:	**60014h-60017h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

Bit	Description
31:29	**Reserved** Project: All Format:
28:16	**Pipe_A_Vertical_Sync_End** Project: All This 13-bit field specifies the Vertical Sync End position expressed in terms of the absolute Line number relative to the vertical active display start. The value programmed should be the VSYNC End line position, where the first active line is considered line 0, the second active line is considered line 1, etc. This register should be loaded with Vactive+BottomBorder+FrontPorch+Sync-1. For interlaced display modes, hardware automatically divides this number by 2 to get the vertical sync end in each field. It does not count the two half lines that get added when operating in modes with half lines.
15:13	**Reserved** Project: All Format:
12:0	**Pipe_A_Vertical_Sync_Start** Project: All This 13-bit field specifies the Vertical Sync Start position expressed in terms of the absolute line number relative to the vertical active display start. The value programmed should be the VSYNC Start line position, where the first active line is considered line 0, the second active line is considered line 1, etc. This register would be loaded with Vactive+BottomBorder+FrontPorch-1. For interlaced display modes, hardware automatically divides this number by 2 to get the vertical sync start in each field. It does not count the two half lines that get added when operating in modes with half lines.

3.1.7 PIPEASRC—Pipe A Source Image Size

PIPEASRC—Pipe A Source Image Size	
Register Type:	MMIO
Address Offset:	**6001Ch-6001Fh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank

Bit	Description
31:28	**Reserved** Project: All Format: MBZ
27:16	**Pipe_A_Horizontal_Source_Image_Size** Project: All This 12-bit field specifies Horizontal source image size up to 4096. This determines the size of the image created by the display planes sent to the blender. The value programmed should be the source image size minus one. It must represent a size that is a multiple of two (even numbers) when driving the LVDS port in two channel mode. This implies that for this mode, the value programmed will always be an odd number. Except in the case of panel fitting internal or in an external device, this register field would be programmed to a value identical to the horizontal active. This is the only register of the timing registers that is allowed to be programmed while the pipe is enabled.
15:12	**Reserved** Project: All Format: MBZ
11:0	**Pipe_A_Vertical_Source_Image_Size** Project: All This 12-bit field specifies the vertical source image size up to 4096 lines. This determines the size of the image created by the display planes sent to the blender. The value programmed should be the source image size minus one. Except in the case of panel fitting internal or in an external device, this register field would be programmed to a value identical to the vertical active. For interlaced display modes, hardware automatically divides this number by 2 to get the vertical source image size in each field.

3.1.8 VSYNCSHIFT_A— Vertical Sync Shift Register

VSYNCSHIFT_A— Vertical Sync Shift Register	
Register Type:	MMIO
Address Offset:	**60028h-6002Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Trusted Type:	1

Bit	Description
31:13	**Reserved** Project: All Format:
12:0	**Pipe_A_Second_Field_Vertical_Sync_Shift** Project: All This value specifies the vertical sync alignment for the start of the interlaced second field expressed in terms of the absolute pixel number relative to the horizontal active display start. This value will only be used if the PIPEACONF is programmed to an interlaced mode. Typically, the interlaced second field vertical sync should start one pixel after the point halfway between successive horizontal syncs, so the value of this register should be programmed to: (horizontal sync start - floor[horizontal total / 2]). (use the actual horizontal sync start and horizontal total values and not the minus one values programmed into the registers) This vertical sync shift only occurs during the interlaced second field. In all other cases the vertical sync start position is aligned with horizontal sync start.

3.1.9 Pipe A M/N Values

Calculation of TU is as follows:

For modes that divide into the link frequency evenly,

Active/TU = payload/capacity

Please note that this is the same ratio as data m/n:

Payload/capacity = dot clk * bytes per pixel / ls_clk * # of lanes

3.1.9.1 PipeADataM1— Pipe A Data M value 1

PipeADataM1— Pipe A Data M value 1			
Register Type: MMIO **Address Offset:** **60030h-60033h** **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 32			
This is the primary pipe data M value used for embedded DisplayPort and FDI. It is used in conjunction with the data N value 1. When switching between two refresh rates, both the M1/N1 data and link values and the M2/N2 link values must be programmed. Data M value 1 is used for the higher power M value setting.			
Bit	**Description**		
31	**Reserved** Project: All	Format: MBZ	
30:25	**TU_Size** This field is the size of the transfer unit for DP, minus one. **Default value to program = 111111 (TU size of 64)**	Project: All	
24	**Reserved** Project: All	Format: MBZ	
23:0	**Pipe_A_Data_M_value** This field is the m value for internal use of the DDA. Calculation of this value is as follows: **Data m/n = dot clock * bytes per pixel / ls_clk * # of lanes** Please note that in the DisplayPort specification, dot clock is referred to as strm_clk In HDMI pixel-repeat modes, the pixel repeat factor needs to be taken into account. The calculation of this value would then be as follows: Data M/N = (dot_clock * bytes_per_pixel) / (ls_clk * num_lanes * pix_repeat_factor)	Project: All	

3.1.9.2 PipeADataN1— Pipe A Data N value 1

PipeADataN1— Pipe A Data N value 1	
Register Type:	MMIO
Address Offset:	**60034h-60037h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

This is the primary pipe data N value used for embedded DisplayPort and FDI. It is used in conjunction with the data M value 1. When switching between two refresh rates, both the M1/N1 data and link values and the M2/N2 link values must be programmed. This value updates at the beginning of vblank. Data N value 1 is used for the higher power N value setting.

Bit	Description
31:24	**Reserved** Project: All Format: MBZ
23:0	**Pipe_A_Data_N_value** Project: All This field is the n value for internal use of the DDA. Calculation of this value is as follows: **Data m/n = dot clock * bytes per pixel / ls_clk * # of lanes** Please note that in the DisplayPort specification, dot clock is referred to as strm_clk In HDMI pixel-repeat modes, the pixel repeat factor needs to be taken into account. The calculation of this value would then be as follows: Data M/N = (dot_clock * bytes_per_pixel) / (ls_clk * num_lanes * pix_repeat_factor)

3.1.9.3 PipeADataM2— Pipe A Data M value 2

PipeADataM2— Pipe A Data M value 2	
Register Type:	MMIO
Address Offset:	**60038h-6003Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

This is the second pipe data M value used for embedded DisplayPort and FDI. It is used in conjunction with the data N value 2. When switching between two refresh rates, both the M1/N1 data and link values and the M2/N2 link values must be programmed. This value updates at the beginning of vblank. Data M value 2 is used for the lower power M value setting.

Bit	Description
31	**Reserved** Project: All Format: MBZ
30:25	**TU_Size** Project: All This field is the size of the transfer unit for DP, minus one. **Default value to program = 111111 (TU size of 64)**
24	**Reserved** Project: All Format: MBZ
23:0	**Pipe_A_Data_M_value** Project: All This field is the m value for internal use of the DDA. Calculation of this value is as follows: **Data m/n = dot clock * bytes per pixel / ls_clk * # of lanes** Please note that in the DisplayPort specification, dot clock is referred to as strm_clk In HDMI pixel-repeat modes, the pixel repeat factor needs to be taken into account. The calculation of this value would then be as follows: Data M/N = (dot_clock * bytes_per_pixel) / (ls_clk * num_lanes * pix_repeat_factor)

3.1.9.4 PipeADataN2— Pipe A Data N value 2

PipeADataN2— Pipe A Data N value 2	
Register Type:	MMIO
Address Offset:	**6003Ch-6003Fh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

This is the second pipe data N value used for DisplayPort and FDI. It is used in conjunction with the data N value 2. When switching between two refresh rates, both the M1/N1 data and link values and the M2/N2 link values must be programmed. This value updates at the beginning of vblank. Data N value 2 is used for the lower power N value setting.

Bit	Description
31:24	**Reserved** Project: All Format: MBZ
23:0	**Pipe_A_Data_N_value** Project: All This field is the n value for internal use of the DDA. Calculation of this value is as follows: **Data m/n = dot clock * bytes per pixel / ls_clk * # of lanes** Please note that in the DisplayPort specification, dot clock is referred to as strm_clk In HDMI pixel-repeat modes, the pixel repeat factor needs to be taken into account. The calculation of this value would then be as follows: Data M/N = (dot_clock * bytes_per_pixel) / (ls_clk * num_lanes * pix_repeat_factor)

3.1.9.5 PipeADPLinkM1— Pipe A Link M value 1

PipeADPLinkM1— Pipe A Link M value 1	
Register Type:	MMIO
Address Offset:	**60040h-60043h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

This is the primary link data M value used for embedded DisplayPort and FDI. It is used in conjunction with the link N value 1. When switching between two refresh rates, both the M1/N1 data and link values and the M2/N2 link values must be programmed. This value updates at the beginning of vblank. Link M value 1 is used for the higher power M value setting.

Bit	Description
31:24	**Reserved** Project: All Format: MBZ
23:0	**Pipe_A_Link_M_value** Project: All This field is the m value for external transmission in the Main Stream Attributes. Calculation of this value is as follows: **Link m/n = pixel clk / ls_clk** Please note that in the DisplayPort specification, pixel clk is referred to as strm_clk In HDMI pixel-repeat modes, the pixel repeat factor needs to be taken into account. The calculation of this value would then be as follows: Link M/N = dot_clock / (ls_clk * pix_repeat_factor)

3.1.9.6 PipeADPLinkN1— Pipe A Link N value 1

PipeADPLinkN1— Pipe A Link N value 1	
Register Type:	MMIO
Address Offset:	**60044h-60047h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

This is the primary link data N value used for embedded DisplayPort and FDI. It is used in conjunction with the link N value 1. When switching between two refresh rates, both the M1/N1 data and link values and the M2/N2 link values must be programmed. This value updates at the beginning of vblank. Link N value 1 is used for the higher power N value setting.

Bit	Description
31:24	**Reserved** Project: All Format: MBZ
23:0	**Pipe_A_Link_N_value** Project: All This field is the n value for external transmission in the Main Stream Attributes and VB-ID. Calculation of this value is as follows (to be filled in): **Link m/n = pixel clk / ls_clk** Please note that in the DisplayPort specification, pixel clk is referred to as strm_clk In HDMI pixel-repeat modes, the pixel repeat factor needs to be taken into account. The calculation of this value would then be as follows: Link M/N = dot_clock / (ls_clk * pix_repeat_factor)

3.1.9.7 PipeADPLinkM2— Pipe A Link M value 2

PipeADPLinkM2— Pipe A Link M value 2	
Register Type:	MMIO
Address Offset:	**60048h-6004Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

This is the secondary link data M value used for embedded DisplayPort and FDI. It is used in conjunction with the link N value 2. When switching between two refresh rates, both the M1/N1 data and link values and the M2/N2 link values must be programmed. This value updates at the beginning of vblank. Link M value 2 is used for the lower power M value setting.

Bit	Description
31:24	**Reserved** Project: All Format: MBZ
23:0	**Pipe_A_Link_M_value** Project: All This field is the m value for external transmission in the Main Stream Attributes. Calculation of this value is as follows: **Link m/n = pixel clk / ls_clk** Please note that in the DisplayPort specification, pixel clk is referred to as strm_clk In HDMI pixel-repeat modes, the pixel repeat factor needs to be taken into account. The calculation of this value would then be as follows: Link M/N = dot_clock / (ls_clk * pix_repeat_factor)

3.1.9.8 PipeADPLinkN2— Pipe A Link N value 2

PipeADPLinkN2— Pipe A Link N value 2	
Register Type:	MMIO
Address Offset:	**6004Ch-6004Fh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

This is the secondary link data N value used for embedded DisplayPort and FDI. It is used in conjunction with the link N value 1. When switching between two refresh rates, both the M1/N1 data and link values and the M2/N2 link values must be programmed. This value updates at the beginning of vblank. Link N value 2 is used for the lower power N value setting.

Bit	Description
31:24	**Reserved** Project: All Format: MBZ
23:0	**Pipe_A_Link_N_value** Project: All This field is the n value for external transmission in the Main Stream Attributes and VB-ID. Calculation of this value is as follows (to be filled in): **Link m/n = pixel clk / ls_clk** Please note that in the DisplayPort specification, pixel clk is referred to as strm_clk In HDMI pixel-repeat modes, the pixel repeat factor needs to be taken into account. The calculation of this value would then be as follows: Link M/N = dot_clock / (ls_clk * pix_repeat_factor)

3.2 Pipe B Timing

3.2.1 HTOTAL_B—Pipe B Horizontal Total Register

HTOTAL_B—Pipe B Horizontal Total Register					
Register Type:	MMIO				
Address Offset:	**61000h-61003h**				
Project:	All				
Default Value:	00000000h				
Access:	R/W				
Size (in bits):	32				
Bit	**Description**				
31:29	**Reserved**	Project:	All	Format:	MBZ
28:16	**Pipe_B_Horizontal_Total_Display_Clocks** See pipe A description			Project:	All
15:12	**Reserved**	Project:	All	Format:	MBZ
11:0	**Pipe_B_Horizontal_Active_Display_Pixels** See pipe A description			Project:	All

3.2.2 HBLANK_B—Pipe B Horizontal Blank Register

HBLANK_B—Pipe B Horizontal Blank Register					
Register Type:	MMIO				
Address Offset:	**61004h-61007h**				
Project:	All				
Default Value:	00000000h				
Access:	R/W				
Size (in bits):	32				
Bit	**Description**				
31:29	**Reserved**	Project:	All	Format:	
28:16	**Pipe_B_Horizontal_Blank_End** See pipe A description			Project:	All
15:13	**Reserved**	Project:	All	Format:	
12:0	**Pipe_B_Horizontal_Blank_Start** See pipe A description			Project:	All

3.2.3 HSYNC_B—Pipe B Horizontal Sync Register

HSYNC_B—Pipe B Horizontal Sync Register	
Register Type:	MMIO
Address Offset:	**61008h-6100Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

Bit	Description
31:29	**Reserved** Project: All Format:
28:16	**Pipe_B_Horizontal_Sync_End** Project: All See pipe A description
15:13	**Reserved** Project: All Format:
12:0	**Pipe_B_Horizontal_Sync_Start** Project: All See pipe A description

3.2.4 VTOTAL_B—Pipe B Vertical Total Register

VTOTAL_B—Pipe B Vertical Total Register	
Register Type:	MMIO
Address Offset:	**6100Ch-6100Fh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

Bit	Description
31:29	**Reserved** Project: All Format:
28:16	**Pipe_B_Vertical_Total_Display_Lines** Project: All See pipe A description
15:12	**Reserved** Project: All Format:
11:0	**Pipe_B_Vertical_Active_Display_Lines** Project: All See pipe A description

3.2.5 VBLANK_B—Pipe B Vertical Blank Register

VBLANK_B—Pipe B Vertical Blank Register					
Register Type:	MMIO				
Address Offset:	**61010h-61013h**				
Project:	All				
Default Value:	00000000h				
Access:	R/W				
Size (in bits):	32				
Bit	**Description**				
31:29	**Reserved**	Project:	All	Format:	
28:16	**Pipe_B_Vertical_Blank_End** See pipe A description			Project:	All
15:13	**Reserved**	Project:	All	Format:	
12:0	**Pipe_B_Vertical_Blank_Start** See pipe A description			Project:	All

3.2.6 VSYNC_B—Pipe B Vertical Sync Register

VSYNC_B—Pipe B Vertical Sync Register					
Register Type:	MMIO				
Address Offset:	**61014h-61017h**				
Project:	All				
Default Value:	00000000h				
Access:	R/W				
Size (in bits):	32				
Bit	**Description**				
31:29	**Reserved**	Project:	All	Format:	
28:16	**Pipe_B_Vertical_Sync_End** See pipe A description			Project:	All
15:13	**Reserved**	Project:	All	Format:	
12:0	**Pipe_B_Vertical_Sync_Start** See pipe A description			Project:	All

3.2.7 PIPEBSRC—Pipe B Source Image Size

PIPEBSRC—Pipe B Source Image Size					
Register Type:	MMIO				
Address Offset:	**6101Ch-6101Fh**				
Project:	All				
Default Value:	00000000h				
Access:	R/W				
Size (in bits):	32				
Double Buffer Update Point:	Start of vertical blank				
Bit	**Description**				
31:28	**Reserved**	Project:	All	Format:	MBZ
27:16	**Pipe_B_Horizontal_Source_Image_Size** See pipe A description			Project:	All
15:12	**Reserved**	Project:	All	Format:	MBZ
11:0	**Pipe_B_Vertical_Source_Image_Size** See pipe A description			Project:	All

3.2.8 VSYNCSHIFT_B— Vertical Sync Shift Register

VSYNCSHIFT_B— Vertical Sync Shift Register					
Register Type:	MMIO				
Address Offset:	**61028h-6102Bh**				
Project:	All				
Default Value:	00000000h				
Access:	R/W				
Size (in bits):	32				
Bit	**Description**				
31:13	**Reserved**	Project:	All	Format:	
12:0	**Pipe_B_Second_Field_Vertical_Sync_Shift** See pipe A description			Project:	All

3.3 Pipe B M/N Values

3.3.1 PipeBDataM1— Pipe B Data M value 1

PipeBDataM1— Pipe B Data M value 1					
Register Type: MMIO **Address Offset:** **61030h-61033h** **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 32					
See pipe A description					
Bit	**Description**				
31	**Reserved**	Project:	All	Format:	MBZ
30:25	**TU_Size** See pipe A description			Project:	All
24	**Reserved**	Project:	All	Format:	MBZ
23:0	**Pipe_B_Data_M_value** See pipe A description			Project:	All

3.3.2 PipeBDataN1— Pipe B Data N value 1

PipeBDataN1— Pipe B Data N value 1					
Register Type: MMIO **Address Offset:** **61034h-61037h** **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 32					
See pipe A description					
Bit	**Description**				
31:24	**Reserved**	Project:	All	Format:	MBZ
23:0	**Pipe_B_Data_N_value** See pipe A description			Project:	All

3.3.3 PipeBDataM2— Pipe B Data M value 2

PipeBDataM2— Pipe B Data M value 2					
Register Type: MMIO **Address Offset:** **61038h-6103Bh** **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 32					
See pipe A description					
Bit	**Description**				
31	**Reserved**	Project:	All	Format:	MBZ
30:25	**TU_Size** See pipe A description			Project:	All
24	**Reserved**	Project:	All	Format:	MBZ
23:0	**Pipe_B_Data_M_value** See pipe A description			Project:	All

3.3.4 PipeBDataN2— Pipe B Data N value 2

PipeBDataN2— Pipe B Data N value 2					
Register Type: MMIO **Address Offset:** **6103Ch-6103Fh** **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 32					
See pipe A description					
Bit	**Description**				
31:24	**Reserved**	Project:	All	Format:	MBZ
23:0	**Pipe_B_Data_N_value** See pipe A description			Project:	All

3.3.5 PipeBDPLinkM1— Pipe B Link M value 1

PipeBDPLinkM1— Pipe B Link M value 1					
Register Type: MMIO **Address Offset:** **61040h-61043h** **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 32					
See pipe A description					
Bit	**Description**				
31:24	**Reserved**	Project:	All	Format:	MBZ
23:0	**Pipe_B_Link_M_value** See pipe A description			Project:	All

3.3.6 PipeBDPLinkN1— Pipe B Link N value 1

PipeBDPLinkN1— Pipe B Link N value 1					
Register Type: MMIO **Address Offset:** **61044h-61047h** **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 32					
See pipe A description.					
Bit	**Description**				
31:24	**Reserved**	Project:	All	Format:	MBZ
23:0	**Pipe_B_Link_N_value** See pipe A description			Project:	All

3.3.7 PipeBDPLinkM2— Pipe B Link M value 2

PipeBDPLinkM2— Pipe B Link M value 2					
Register Type: MMIO **Address Offset:** **61048h-6104Bh** **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 32					
See pipe A description					
Bit	**Description**				
31:24	**Reserved**	Project:	All	Format:	MBZ
23:0	**Pipe_B_Link_M_value** See pipe A description			Project:	All

3.3.8 PipeBDPLinkN2— Pipe B Link N value 2

PipeBDPLinkN2— Pipe B Link N value 2					
Register Type: MMIO **Address Offset:** **6104Ch-6104Fh** **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 32					
See pipe A description					
Bit	**Description**				
31:24	**Reserved**	Project:	All	Format:	MBZ
23:0	**Pipe_B_Link_N_value** See pipe A description			Project:	All

3.4 Embedded DP CTL & Aux Channel

3.4.1 DP_A—DisplayPort A Control Register

DP_A—DisplayPort A Control Register

Register Type: MMIO
Address Offset: 64000h
Project: All
Default Value: 00000018h
Access: R/W
Size (in bits): 32

Bit	Description
31	**DisplayPort_A_Enable** Project: All Default Value: 0b Disabling this port will put it in its lowest power state. Port enable takes place on the Vblank after being written. [DevSNB] If DisplayPort A and another port have been both enabled, then prior to disabling DisplayPort A the FDI PLL must be enabled in either FDI_TXA_CTL or FDI_TXB_CTL, then DisplayPort A can be disabled, and then the FDI PLL can be restored to its previous state.

Value	Name	Description	Project
0b	Disable	Disable and tristates the Display Port A interface	All
1b	Enable	Enable. This bit enables the Display Port A interface.	All

Bit	Description
30	**Pipe_Select** Project: All Default Value: 0b This bit determines from which display pipe the source data will originate. Pipe selection takes place on the Vblank after being written

Value	Name	Description	Project
0b	Pipe A	Pipe A	All
1b	Pipe B	Pipe B	All

DP_A—DisplayPort A Control Register

29:28

Link_training_pattern_enable

Project: All

Default Value: 0b

These bits are used for link initialization as defined in the DisplayPort specification. Please note that the link must first be configured prior to sending training patterns.

When enabling the port, it must be turned on with pattern 1 enabled. When retraining a link, the port must be disabled, then re-enabled with pattern 1 enabled.

Value	Name	Description	Project
00b	P1	**Pattern 1 enabled:** Repetition of D10.2 characters	All
01b	P2	**Pattern 2 enabled:** Repetition of K28.5, D11.6, K28.5, D11.6, D10.2, D10.2, D10.2, D10.2, D10.2, D10.2. Please note that the entire pattern must complete before another pattern is sent. Scrambling initialization and disparity init commence at the end of the last iteration of pattern 2.	All
10b	Idle	**Idle Pattern enabled:** Transmit BS followed by VB-ID with NoVideoStream_flag set to 1, five times	
11b	None	**Link not in training:** Send normal pixels	

27:22

Voltage_swing_Pre-emphasis_level_set_SNB

Project: DevSNB

Default Value: 000000b

These bits are used for setting link voltage swing and pre-emphasis.

Value	Name	Description	Project
000000b	400mv_0db/ 600mv_0db	400mv 0dB / 600mv 0dB Actual 0.5v transition and 0.5v non-transition	DevSNB
000001b	400mv_3.5db	400mv 3.5dB Actual 0.5v transition and 0.33v non-transition	DevSNB
111010b	400mv 6db / 600mv_6db	400mv 6dB / 600mv 6dB Actual 1v transition and 0.5v non-transition	DevSNB
111001b	600mv 3.5db / 800mv_3.5db	600mv 3.5dB / 800mv 3.5dB Actual 1v transition and 0.66v non-transition	DevSNB
111000b	800mv_0db / 1200mv_0db	800mv 0dB / 1200mv 0dB Actual 1v transition and 1v non-transition	DevSNB
Others	Reserved	Reserved	All

DP_A—DisplayPort A Control Register

21:19

Port_Width_Selection

Project: All

Default Value: 0b

This bit selects the number of lanes to be enabled on the DisplayPort link. Port width change must be done as a part of mode set.

Value	Name	Description	Project
000b	x1	x1 Mode (Default)	All
001b	x2	x2 Mode.	All
010b	Reserved	Reserved	All
011b	x4	x4 Mode.	All
1XXb	Reserved	Reserved	All

18

Enhanced_Framing_Enable

Project: All

Default Value: 0b

This bit selects enhanced framing.

Locked once port is enabled. Updates when the port is disabled then re-enabled

Value	Name	Description	Project
0b	Disable	Enhanced framing disabled	All
1b	Enable	Enhanced framing enabled.	All

17:16

DP_PLL_Frequency_Select

Project: All

Default Value: 0b

Value	Name	Description	Project
00b	270mhz	270mhz	All
01b	162mhz	162mhz.	All
1Xb	Reserved	Reserved	All

DP_A—DisplayPort A Control Register

15 — Port_reversal

Project: All

Default Value: 0b

Enables lane reversal within the port: lane 0 mapped to lane 3, lane 1 mapped to lane 2, etc. Port reversal does not affect AUX channel lane mapping.

Locked once port is enabled. Updates when the port is disabled then re-enabled

Value	Name	Description	Project
0b	Normal	Port not reversed	All
1b	Reversed	Port reversed	All

14 — DP_PLL_enable

Project: All

Default Value: 0b

This bit enables the DP PLL. Please note that software must wait for the PLL warmup cycle before enabling the port through bit 31 of this register.

[DevILK] Work around required when enabling DP PLL while a pipe is enabled going to FDI

1. Wait for the start of vertical blank on the enabled pipe going to FDI
2. Program DP PLL enable

Value	Name	Description	Project
0b	Disable	**DP PLL not enabled**	All
1b	Enable	**DP PLL enabled**	All

13:8 — Reserved Project: All Format: MBZ

7 — Reserved

6 — Reserved Project: All Format: MBZ

5 — Reserved

4:3 — Sync_Polarity

Project: All

Default Value: 11b VS and HS are active high

Indicates the polarity of Hsync and Vsync to be transmitted in MSA.

Value	Name	Description	Project
00b	Low	VS and HS are active low (inverted)	All
01b	VS_Low, HS_High	VS is active low (inverted), HS is active high	All
10b	VS_High, HS_Low	VS is active high, HS is active low (inverted)	All
11b	High	VS and HS are active high	

DP_A—DisplayPort A Control Register	
2	**Digital_Display_A_Detected** Project: All Default Value: 0b Read-only bit indicating whether a digital display was detected during initialization. This bit is qualified with the Embedded DisplayPort A capability fuse.

Value	Name	Description	Project
0b	No Detect	Digital display not detected during initialization	All
1b	Detected	Digital display detected during initialization	All

1:0	**Reserved** Project: All Format: MBZ

3.4.2 DPA_AUX_CH_CTL—Display Port A AUX Channel Control

DPA_AUX_CH_CTL—Display Port A AUX Channel Control

Register Type: MMIO
Address Offset: **64010h-64013h**
Project: All
Default Value: 00050000h
Access: R/W Special
Size (in bits): 32

Bit	Description
31	**Send/Busy** Project: All Format: Setting this bit to a one initiates the transaction, when read this bit will be a 1 until the transmission completes. The transaction is completed when the response is received or when a timeout occurs. Do not write a 1 again until transaction completes. Writes of 0 will be ignored. **Programming Notes** Do not change any fields while Busy bit 31 is asserted.
30	**Done** Project: All Access: R/W Clear A sticky bit that indicates the transaction has completed. SW must write a 1 to this bit to clear the event
29	**Interrupt_on_Done** Project: All Format: Enable an interrupt in the hotplug status register when the transaction completes or times out.
28	**Time_out_error** Project: All Access: R/W Clear A sticky bit that indicates the transaction has timed out. SW must write a 1 to this bit to clear the event.

DPA_AUX_CH_CTL—Display Port A AUX Channel Control

27:26 **Time_out_timer_value**

Project: All

Default Value: 0b

The time count depends on the 2X bit clock divider (bits 10:0) being programmed for 2MHz.

Value	Name	Description	Project
00b	400us	400us	All
01b	600us	600us	All
10b	800us	800us	All
11b	1600us	1600us	All

25 **Receive_error** Project: All

Access: R/W Clear

Default Value: 0b

A sticky bit that indicates that the data received was corrupted, not in multiples of a full byte, or more than 20 bytes. SW must write a 1 to this bit to clear the event.

24:20 **Message_Size** Project: All Format:

This field is used to indicate the total number bytes to transmit (including the header). It also indicates the number of bytes received in a transaction (including the header). This field is valid only when the done bit is set and timeout or receive error has not occurred. Sync/Stop are not part of the message or the message size.

Reads of this field will give the response message size.

The read value will not be valid while Busy bit 31 is asserted.

Message sizes of 0 or >20 are not allowed.

19:16 **Precharge_Time** Project: All

Default Value: 0101b 10 us precharge

Used to determine the precharge time for the Aux Channel drivers.

The value is the number of microseconds times 2 (assuming 2X bit clock divider programmed for 2MHz).

Example:

For 10us precharge, program 5 (10us/2us).

15:11 **Reserved**

10:0 **2X_Bit_Clock_divider** Project: All

Used to determine the 2X bit clock the Aux Channel logic runs on.

This value divides the input clock frequency down to 2X bit clock rate. The 2X bit clock rate is ideally 2MHz (0.5us). The input clock is cdclk.

Example:

For 300MHz input clock and desired 2MHz 2X bit clock, program 150 (300MHz/2MHz).

3.4.3 DPA_AUX_CH_DATA1—Display Port A AUX Data Register 1

DPA_AUX_CH_DATA1—Display Port A AUX Data Register 1	
Register Type:	MMIO
Address Offset:	**64014h-64017h**
Project:	All
Default Value:	00000000h
Access:	R/W Special
Size (in bits):	32
The read value will not be valid while Busy bit 31 is asserted.	

Bit	Description
31:0	**AUX_CH_DATA1** Project: All Format: The first DWord of the message. The MSbyte is transmitted first. Reads will give the response data after transaction complete.

3.4.4 DPA_AUX_CH_DATA2—Display Port A AUX Data Register 2

DPA_AUX_CH_DATA2—Display Port A AUX Data Register 2	
Register Type:	MMIO
Address Offset:	**64018h-6401Bh**
Project:	All
Default Value:	00000000h
Access:	R/W Special
Size (in bits):	32
The read value will not be valid while Busy bit 31 is asserted.	

Bit	Description
31:0	**AUX_CH_DATA2** Project: All Format: The second DWord of the message. The MSbyte is transmitted first. Only used if the message size is greater than 4. Reads will give the response data after transaction complete

3.4.5 DPA_AUX_CH_DATA3—Display Port A AUX Data Register 3

DPA_AUX_CH_DATA3—Display Port A AUX Data Register 3	
Register Type:	MMIO
Address Offset:	**6401Ch-6401Fh**
Project:	All
Default Value:	00000000h
Access:	R/W Special
Size (in bits):	32

The read value will not be valid while Busy bit 31 is asserted.

Bit	Description
31:0	**AUX_CH_DATA3** Project: All Format: The third DWord of the message. The MSbyte is transmitted first. Only used if the message size is greater than 8. Reads will give the response data after transaction complete.

3.4.6 DPA_AUX_CH_DATA4—Display Port A AUX Data Register 4

DPA_AUX_CH_DATA4—Display Port A AUX Data Register 4	
Register Type:	MMIO
Address Offset:	**64020h-64023h**
Project:	All
Default Value:	00000000h
Access:	R/W Special
Size (in bits):	32

The read value will not be valid while Busy bit 31 is asserted.

Bit	Description
31:0	**AUX_CH_DATA4** Project: All Format: The fourth DWord of the message. The MSbyte is transmitted first. Only used if the message size is greater than 12. Reads will give the response data after transaction complete.

3.4.7 DPA_AUX_CH_DATA5—Display Port A AUX Data Register 5

DPA_AUX_CH_DATA5—Display Port A AUX Data Register 5	
Register Type:	MMIO
Address Offset:	**64024h-64027h**
Project:	All
Default Value:	00000000h
Access:	R/W Special
Size (in bits):	32

The read value will not be valid while Busy bit 31 is asserted.

Bit	Description
31:0	**AUX_CH_DATA5** Project: All Format: The fifth DWord of the message. The MSbyte is transmitted first. Only used if the message size is greater than 16. Reads will give the response data after transaction complete.

3.5 Panel Fitter Control Registers

3.5.1 PF_WIN_POS—Panel Fitter Window Position

PFA_WIN_POS—Panel Fitter A Window Position	
Register Type:	MMIO
Address Offset:	**68070h-68073h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to PFA_WIN_SZ

Bit	Description
31:29	**Reserved** Project: All Format: MBZ
28:16	**XPOS** Project: All The X coordinate (in pixels) of the upper left most pixel of the panel fitted display window.
15:12	**Reserved** Project: All Format: MBZ
11:0	**YPOS** Project: All The Y coordinate (in lines) of the upper left most pixel of the panel fitter display window. LSB must be zero for interlaced modes.

PFB_WIN_POS—Panel Fitter B Window Position	
Register Type:	MMIO
Address Offset:	**68870h-68873h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to PFB_WIN_SZ

Bit	Description				
31:29	**Reserved**	Project:	All	Format:	MBZ
28:16	**XPOS** See PFA description	Project:	All		
15:12	**Reserved**	Project:	All	Format:	MBZ
11:0	**YPOS** See PFA description	Project:	All		

3.5.2 PF_WIN_SZ—Panel Fitter Window Size

Coordinates are determined with a value of (0,0) being the upper left corner of the display device (rotation does not affect this). Writes to the window size arm PF registers for the pipe.

PFA_WIN_SZ—Panel Fitter A Window Size	
Register Type:	MMIO
Address Offset:	**68074h-68077h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank

Bit	Description				
31:29	**Reserved**	Project:	All	Format:	MBZ
28:16	**XSIZE** The horizontal size in pixels of the desired panel fitted window.	Project:	All		
15:12	**Reserved**	Project:	All	Format:	MBZ
11:0	**YSIZE** The vertical size in pixels of the desired panel fitted window. LSB must be zero for interlaced modes	Project:	All		

PFB_WIN_SZ—Panel Fitter B Window Size	
Register Type:	MMIO
Address Offset:	68874h
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank

Bit	Description
31:29	**Reserved** Project: All Format: MBZ
28:16	**XSIZE** Project: All See PFA descriptio
15:12	**Reserved** Project: All Format: MBZ
11:0	**YSIZE** Project: All See PFA description

3.5.3 PF_CTRL_1—Panel Fitter Control 1

PFA_CTRL_1—Panel Fitter A Control 1	
Register Type:	MMIO
Address Offset:	**68080h-68083h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to PFA_WIN_SZ

When using panel fitter downscaling (pipe source size is larger than panel fitter window size) the maximum supported pixel rate will be reduced by the downscale amount.

Bit	Description
31	**Enable_Pipe_Scaler** Project: All Default Value: 0b

Value	Name	Description	Project
0b	Disable	Data bypasses the scaler	All
1b	Enable	The scaler is enabled	All

Bit	Description
30	**Reserved**
29	**Reserved**
28	**Reserved**

PFA_CTRL_1—Panel Fitter A Control 1

27	**VADAPT** Project: All Puts the adaptive vertical filter into adaptive mode, intended for use in interlace output modes only.

Value	Name	Description	Project
0b	Disable	Adaptive filtering disabled	All
1b	Enable	Adaptive filtering enabled	All

26:25	**VADAPT_MODE** Project: All Puts the adaptive vertical filter into adaptive mode, intended for use in interlace output modes only.

Value	Name	Description	Project
00b	Least Adaptive	Least Adaptive (Recommended)	All
01b	Moderately Adaptive	Moderately Adaptive	All
10b	Reserved	Reserved	All
11b	Most Adaptive	Most Adaptive	All

24:23	**FILTER_SELECT** Project: All Filter coefficient selection

Value	Name	Description	Project
00b	Programmed	Programmed Coefficients (Recommended)	All
01b	Hardcoded Med	Hardcoded Coefficients for Medium 3x3 Filtering	All
10b	Edge Enhance	Hardcoded Coefficients for Edge Enhancing 3x3 Filtering	All
11b	Edge Soften	Hardcoded Coefficients for Edge Softening 3x3 Filtering	All

22	**CHR_PREF** Project: All Chroma Pre-filter enable.

Value	Name	Description	Project
0b	Disable	Pre-filter disabled	All
1b	Enable	Pre-filter enabled	All

21	**Reserved**
20	**Reserved**
19:0	**Reserved** Project: All Format: MBZ

PFB_CTRL_1—Panel Fitter B Control 1

Register Type:	MMIO
Address Offset:	**68880h-68883h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to PFB_WIN_SZ

When using panel fitter downscaling (pipe source size is larger than panel fitter window size) the maximum supported pixel rate will be reduced by the downscale amount.

Bit	Description
31	**Enable_Pipe_Scaler** Project: All Default Value: 0b

Value	Name	Description	Project
0b	Disable	Data bypasses the scaler	All
1b	Enable	The scaler is enabled	All

Bit	Description
30	**Reserved**
29	**Reserved**
28	**Reserved**
27	**VADAPT** Project: All Puts the adaptive vertical filter into adaptive mode, intended for use in interlace output modes only.

Value	Name	Description	Project
0b	Disable	Adaptive filtering disabled	All
1b	Enable	Adaptive filtering enabled	All

Bit	Description
26:25	**VADAPT_MODE** Project: All Puts the adaptive vertical filter into adaptive mode, intended for use in interlace output modes only.

Value	Name	Description	Project
00b	Least Adaptive	Least Adaptive (Recommended)	All
01b	Moderately Adaptive	Moderately Adaptive	All
10b	Reserved	Reserved	All
11b	Most Adaptive	Most Adaptive	All

PFB_CTRL_1—Panel Fitter B Control 1	
24:23	**FILTER_SELECT** Project: All Filter coefficient selection

Value	Name	Description	Project
00b	Programmed	Programmed Coefficients (Recommended)	All
01b	Hardcoded Med	Hardcoded Coefficients for Medium 3x3 Filtering	All
10b	Edge Enhance	Hardcoded Coefficients for Edge Enhancing 3x3 Filtering	All
11b	Edge Soften	Hardcoded Coefficients for Edge Softening 3x3 Filtering	All

22	**CHR_PREF** Project: All Chroma Pre-filter enable.

Value	Name	Description	Project
0b	Disable	Pre-filter disabled	All
1b	Enable	Pre-filter enabled	All

21	**Reserved**
20	**Reserved**
19:0	**Reserved** Project: All Format: MBZ

3.5.4 Panel Fitter Coefficient Registers

Coefficients for the panel fitter filters are stored in sign-exponent-mantissa format. The number of mantissa bit varies based on the filter. Two filter coefficients are stored in each dword, the tables below show the data packing in each of the words. Unused bits are considered reserved and should be written zero. The default value of all coefficient registers is 00000000h. Coefficients greater than 1.0 are only allowed in the center tap of the filter.

For RGB modes the Luma and Chroma filter coeffs are programmed with the same values.

Panel Fitter Coefficient Definition	
Project: All	
Bit	**Description**
15	**Sign_bit** Project: All

Value	Name	Description	Project
0b	Positive	Positive	All
1b	Negative	Negative	All

14	**Reserved** Project: All Format: MBZ

Panel Fitter Coefficient Definition	
13:12	**Exponent_bits** Project: All The meaning of the exponent bits varies for center tap or non-center tap coefficients.

Value	Name	Description	Project
00b	2 or 0.125	Center taps: 2 or mantissa is b.bbbbbb… Non-center taps: 0.125 or mantissa is 0.000bbbbbbb	All
01b	1	1 or mantissa is 0.bbbbbbb…	All
10b	0.5	0.5 or mantissa is 0.0bbbbbbb…	All
11b	0.25	0.25 or mantissa is 0.00bbbbbbb…	All
others	Reserved	Reserved	All

11:3	**Mantissa** Project: All Size of the mantissa varies based on the filter, but the MSB of the mantissa is always bit 11. Center tap coefficients use all 9 bits of mantissa. Non-center tap coefficients use only the upper 7 bits of mantissa and the lower 2 bits are ignored.
2:0	**Reserved** Project: All Format: MBZ

3.6 Panel Fitter Horizontal Coefficients

Coefficients are packed in the horizontal coefficient registers as follows (with the letter representing the tap and the number representing the coefficient set):

Address	bits [31:16]	bits[15:0]
68x00	B0	A0
68x04	D0	C0
68x08	F0	E0
68x0C	A1	G0
68x10	C1	B1

etc….

3.6.1 PF_HFILTL_COEF—Panel Fitter Horizontal Luma/Red Coefficients

PFA_HFILTL_COEF—Panel Fitter A Horizontal Luma/Red Coefficients

Register Type: MMIO
Address Offset: **68100h-681EFh**
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 60x32

17 phases of 7 taps require 60 dwords

Center coefficient is 1.2.9

Other coefficients are 1.2.7

DWord	Bit	Description				
0..59	31:16	**Coefficient2**	Project:	All	Format:	Panel Fitter Coefficient Definition
	15:0	**Coefficient1**	Project:	All	Format:	Panel Fitter Coefficient Definition

PFB_HFILTL_COEF—Panel Fitter B Horizontal Luma/Red Coefficients

Register Type: MMIO
Address Offset: **68900h-689EFh**
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 60x32

17 phases of 7 taps require 60 dwords

Center coefficient is 1.2.9

Other coefficients are 1.2.7

DWord	Bit	Description				
0..59	31:16	**Coefficient2**	Project:	All	Format:	Panel Fitter Coefficient Definition
	15:0	**Coefficient1**	Project:	All	Format:	Panel Fitter Coefficient Definition

3.6.2 PF_HFILTC_COEF—Panel Fitter Horizontal Chroma/Green and Blue Coefficients

PFA_HFILTC_COEF—Panel Fitter A Horizontal Chroma/Green and Blue Coefficients						
Register Type:	MMIO					
Address Offset:	**68200h-682EFh**					
Project:	All					
Default Value:	00000000h					
Access:	R/W					
Size (in bits):	60x32					
17 phases of 7 taps require 60 dwords Center coefficient is 1.2.9 Other coefficients are 1.2.7						
DWord	**Bit**	**Description**				
0..59	31:16	**Coefficient2**	Project:	All	Format:	Panel Fitter Coefficient Definition
	15:0	**Coefficient1**	Project:	All	Format:	Panel Fitter Coefficient Definition

PFB_HFILTC_COEF—Panel Fitter B Horizontal Chroma/Green and Blue Coefficients						
Register Type:	MMIO					
Address Offset:	**68A00h-68AEFh**					
Project:	All					
Default Value:	00000000h					
Access:	R/W					
Size (in bits):	60x32					
17 phases of 7 taps require 60 dwords Center coefficient is 1.2.9 Other coefficients are 1.2.7						
DWord	**Bit**	**Description**				
0..59	31:16	**Coefficient2**	Project:	All	Format:	Panel Fitter Coefficient Definition
	15:0	**Coefficient1**	Project:	All	Format:	Panel Fitter Coefficient Definition

3.7 Panel Fitter Vertical Coefficients

Coefficients are packed in the vertical coefficient registers as follows (with the letter representing the tap and the number representing the coefficient set). When the vertical filter is in 3 line mode the three taps used are A, C & E, B & C must be programmed to zero in three line mode.

Address	bits [31:16]	bits[15:0]
68x00	B0	A0
68x04	D0	C0
68x08	A1	E0
68x0C	C1	B1
68x10	E1	D1

etc….

3.7.1 PF_VFILTL_COEF—Panel Fitter Vertical Luma/Red Coefficients

PFA_VFILTL_COEF—Panel Fitter A Vertical Luma/Red Coefficients

Register Type: MMIO
Address Offset: **68300h-683ABh**
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 43x32

17 phases of 5 taps require 43 dwords

Center coefficient is 1.2.9

Other coefficients are 1.2.7

DWord	Bit	Description				
0..42	31:16	**Coefficient2**	Project:	All	Format:	Panel Fitter Coefficient Definition
	15:0	**Coefficient1**	Project:	All	Format:	Panel Fitter Coefficient Definition

PFB_VFILTL_COEF—Panel Fitter B Vertical Luma/Red Coefficients						
Register Type:	MMIO					
Address Offset:	**68B00h-68BABh**					
Project:	All					
Default Value:	00000000h					
Access:	R/W					
Size (in bits):	43x32					
17 phases of 5 taps require 43 dwords Center coefficient is 1.2.9 Other coefficients are 1.2.7						
DWord	**Bit**	**Description**				
0..42	31:16	**Coefficient2**	Project:	All	Format:	Panel Fitter Coefficient Definition
	15:0	**Coefficient1**	Project:	All	Format:	Panel Fitter Coefficient Definition

3.7.2 PF_VFILTC_COEF—Panel Fitter Vertical Chroma/Green and Blue Coefficients

PFA_VFILTC_COEF—Panel Fitter A Vertical Chroma/Green and Blue Coefficients						
Register Type:	MMIO					
Address Offset:	**68400h-684ABh**					
Project:	All					
Default Value:	00000000h					
Access:	R/W					
Size (in bits):	43x32					
17 phases of 5 taps require 43 dwords Center coefficient is 1.2.9 Other coefficients are 1.2.7						
DWord	**Bit**	**Description**				
0..42	31:16	**Coefficient2**	Project:	All	Format:	Panel Fitter Coefficient Definition
	15:0	**Coefficient1**	Project:	All	Format:	Panel Fitter Coefficient Definition

PFB_VFILTC_COEF—Panel Fitter B Vertical Chroma/Green and Blue Coefficients

Register Type: MMIO
Address Offset: **68C00h-68CABh**
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 43x32

17 phases of 5 taps require 43 dwords

Center coefficient is 1.2.9

Other coefficients are 1.2.7

DWord	Bit	Description				
0..42	31:16	**Coefficient2**	Project:	All	Format:	Panel Fitter Coefficient Definition
	15:0	**Coefficient1**	Project:	All	Format:	Panel Fitter Coefficient Definition

4. Plane Controls (70000h–7FFFFh)

4.1 Display Pipeline A

4.1.1 PIPEA_DSL—Pipe A Display Scan Line

PIPEA_DSL—Pipe A Display Scan Line

Register Type: MMIO
Address Offset: **70000h-70003h**
Project: All
Default Value: 00000000h
Access: Read Only
Size (in bits): 32

This register enables the read back of the display pipe vertical "line counter". The value increments at the leading edge of HSYNC and can be safely read any time. The value resets to line zero at the first active line of the display. In interlaced display timings, the scan line counter provides the current line in the field. One field will have a total number of lines that is one greater than the other field. Content locked display will adjust the total number of lines displayed.

Bit	Description
31	**Current_Field** Project: All Default Value: 0 Provides read back of the current field being displayed on display pipe A.
30:13	**Reserved** Project: All Format:
12:0	**Line_Counter_for_Display** Project: All Provides read back of the display pipe A vertical line counter. This is an indication of the current display scan line to be used by software to synchronize with the display.

Value	Name	Description	Project
0b	Odd	First field (odd field)	All
1b	Even	Second field (even field)	All

4.1.2 PIPEA_SLC—Pipe A Display Scan Line Count Range Compare

PIPEA_SLC—Pipe A Display Scan Line Count Range Compare

Register Type: MMIO
Address Offset: **70004h-70007h**
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 32

[DevSNB:D2+] Reserved. The waid for scan line is programmed through DE_LOAD_SEL 0x4F100.

[DevSNB:D2] The scan line number register is compared with the display line value from the pipe timing generator. The result of this comparison is used to generate interrupts and render responses.
The value programmed should be desired value – 1, so for line 0, the value programmed is VTOTAL, and for line 1, the value programmed is 0. Content locked display will adjust the total number of lines displayed.

Bit	Description
31	**Inclusive/Exclusive** Project: **DevILK** Default Value: 0b

Value	Name	Description	Project
0b	Exclusive	Exclusive: outside of the range	All
1b	Inclusive	Inclusive: within the range	All

Bit	Description
30:29	**Reserved** Project: **DevILK** Format: MBZ
28:16	**Start_Scan_Line_Number** Project: **DevILK** Format: Range: 0..Vertical Total This field specifies the starting scan line number of the Scan Line Window. Scan line 0 is the first line of the display frame.
15:13	**Reserved** Project: **DevILK** Format: MBZ
12:0	**End_Scan_Line_Number** Project: **DevILK** Format: Range: 0..Vertical Total This field specifies the ending scan line number of the Scan Line Window. Scan line 0 is the first line of the display frame.
31:13	**Reserved** Project: **DevSNB** Format: MBZ
12:0	**Scan_Line_Number** Project: **Pre-DevSNB:D2** Range 0..Vertical Total This field specifies the scan line number on which to generate scan line interrupt and render response.
12:0	**Reserved** Project: **DevSNB:D2+** Format: MBZ

4.1.3 PIPEACONF—Pipe A Configuration Register

PIPEACONF—Pipe A Configuration Register	
Register Type:	MMIO
Address Offset:	**70008h-7000Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank OR pipe disabled

Bit	Description
31	**Pipe_A_Enable** Project: All Default Value: 0b Setting this bit to the value of one, turns on pipe A. This must be done before any planes are enabled on this pipe. Changing it to a zero should only be done when all planes that are assigned to this pipe have been disabled. Turning the pipe enable bit off disables the timing generator in this pipe. Synchronization pulses to the display are not maintained if the timing generator is disabled. Pipe timing registers must contain valid values before this bit is enabled.

Value	Name	Description	Project
0b	Disable	Disable	All
1b	Enable	Enable	All

Bit	Description
30	**Pipe_State** Project: All Default Value: 0b This bit indicates the actual state of the pipe. Since there can be some delay between disabling the pipe and the pipe actually shutting off, this bit indicates the true current state of the pipe.

Value	Name	Description	Project
0b	Disable	Disable State	All
1b	Enable	Enable State	All

Bit	Description
29	**Reserved** Project: All
28:27	**Reserved**
26	**Reserved** Project: DevSNB

PIPEACONF—Pipe A Configuration Register

25:24

Pipe_A_Palette/Gamma_Unit_Mode

Project: All

Default Value: 0b

These bits select which mode the pipe gamma correction logic works in. See the Display Palette Registers for information on the different palette/gamma modes. Other gamma units such as in the sprite are unaffected by this bit.

Value	Name	Description	Project
00b	8 bit	8-bit Legacy Palette Mode	All
01b	10 bt	10-bit Precision Palette Mode	All
10b	12 bit	12-bit Interpolated Gamma Mode	All
11b	Reserved	Reserved	All

23:21

Interlaced_Mode

Project: All

Default Value: 0b

These bits are used for software control of interlaced behavior. They are updated immediately if the pipe is off, or in the vertical blank after programming if pipe is enabled.

Note: VGA display modes do not work while in interlaced fetch modes

Value	Name	Description	Project
000b	PF-PD	Progressive Fetch / Progressive display	All
001b	PF-ID	Progressive Fetch / Interlaced display (HDMI) **Requires panel fitting to be enabled**	All
010b	Reserved	Reserved	All
011b	IF-ID	Interlaced Fetch / Interlaced display (programmable sync, normal interlaced)	All
100b	IF-ID-DBL	Interlaced embedded panel with interlaced fetch (pixel doubling power savings mode, no PF enabled)	All
101b	PF-ID-DBL	Interlaced embedded panel with progressive fetch (pixel doubling power savings mode with PF enabled) **Requires panel fitting to be enabled**	All
others	Reserved	Reserved	All

PIPEACONF—Pipe A Configuration Register

20 — **Display_Power_Mode_Switch**

Project: All

Default Value: 0b

This bit is used for software to set the power saving progressive mode. The pipe enters or exits the power savings mode on the vblank after this bit is written. Please note that bits 17:16 of this register must be set to 00 in order for this bit to take effect.

If LVDS clockgating is disabled (bits 14, 30 set to '1' in 0xC2020), then clockgating must be temporarily enabled (bits 14, 30 cleared to '0') when toggling Display Power Mode Switch followed by wait of 2 vblanks and then disabled again.

Value	Name	Description	Project
0b	Progressive	Pipe is in progressive mode	All
1b	Power save	Pipe is in power savings progressive mode	All

19:18 — **MSA_Timing_Delay**

Project: DevSNB

Default Value: 00b

This field selects the vertical blank line on which MSA is sent. It is intended for use with embedded DisplayPort panels that support sDRRS. The sDRRS timing switch shall occur on same line as the MSA.

Value	Name	Description	Project
00b	Line1	MSA and sDRRS timing switch occur within the first line of vertical blank	All
01b	Line2	MSA and sDRRS timing switch occur within the second line of vertical blank	All
10b	Line3	MSA and sDRRS timing switch occur within the third line of vertical blank	All
11b	Line4	MSA and sDRRS timing switch occur within the fourth line of vertical blank	All

19:18 — **Reserved** Project: DevILK Format: MBZ

17:16 — **Reserved** Project: All Format: MBZ

15:14 — **Display_Rotation_Info**

Project: All

Default Value: 0b

These are informative bits set by software to indicate this pipe is being rotated. Software should set these for both hardware and software rotation cases. Hardware rotation is **not** enabled through these bits.

Value	Name	Description	Project
00b	None	No rotation on this pipe	All
01b	90	90° rotation on this pipe	All
10b	180	180° rotation on this pipe	All
11b	270	270° rotation on this pipe	All

PIPEACONF—Pipe A Configuration Register

13 — **Color_Range_Select**

Project: All

Default Value: 0b

This bit is used to select the color range of outputs.

Value	Name	Description	Project
0b	Full	Apply full 0-2^n - 1 color range to the output	All
1b	CE	Apply CE color range to the output	All

12:11 — **Pipe_output_color_space_select**

Project: All

Default Value: 0b

Informs the ports of the pipe output color space. Plane data formats and CSC need to be programmed to match what is selected here.

Value	Name	Description	Project
00b	RGB	RGB	All
01b	YUV 601	YUV 601	All
10b	YUV 709	YUV 709	All
11b	Reserved	Reserved	All

10:9 — **Reserved** Project: DevSNB

8 — **Reserved** Project: All Format: MBZ

7:5 — **Bits_Per_Color**

Project: All

Default Value: 0b

This field selects the number of bits per color sent to a receiver device connected to this pipe. Color format takes place on the Vblank after being written. Color format change can be done independent of a pixel clock change.

Software should enable dithering in the pipe/port if selecting a pixel color depth higher or lower than the pixel color depth of the frame buffer.

Value	Name	Description	Project
000b	8 bits	8 bits per color	All
001b	10 bits	10 bits per color	All
010b	6 bits	6 bits per color	All
011b	12 bits	12 bits per color	All
1XXb	Reserved	Reserved	All

PIPEACONF—Pipe A Configuration Register

4	**Dithering_enable** Project: All Default Value: 0b This bit enables dithering

Value	Name	Description	Project
0b	Disable	Dithering disabled	All
1b	Enable	Dithering enabled	All

3:2	**Dithering_type** Project: All Default Value: 0b These bits select dithering type.

Value	Name	Description	Project
00b	Spatial	Spatial only	All
01b	ST1	Spatio-Temporal 1	All
10b	ST2	Spatio-Temporal 2 (testmode)	All
11b	Temporal	Temporal only (testmode)	All

1	**Reserved**
0	**Reserved** Project: All Format: MBZ

4.2 Display Pipeline A Counters and Timestamps

4.2.1 PIPEA_FRMCOUNT—Pipe A Frame Counter

PIPEA_FRMCOUNT—Pipe A Frame Counter	
Register Type:	MMIO
Address Offset:	**70040h-70043h**
Project:	All
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32
Bit	**Description**
31:0	**Pipe_Frame_Counter** Project: All Format: Provides read back of the display pipe frame counter. This counter increments on every start of vertical blank and rolls over back to 0 after 2^32 frames.

4.2.2 PIPEA_FLIPCOUNT—Pipe A Flip Counter

PIPEA_FLIPCOUNT—Pipe A Flip Counter	
Register Type:	MMIO
Address Offset:	**70044h-70047h**
Project:	All
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32
Bit	**Description**
31:0	**Pipe_Flip_Counter** Project: All Format: Provides read back of the display pipe flip counter. This counter increments on each flip of the surface of the primary plane on this pipe. This includes command streamer asynchronous and synchronous flips and any MMIO writes to the primary plane surface address. It rolls over back to 0 after 2^32 flips.

4.2.3 PIPEA_FRMTIMESTAMP—Pipe A Frame Time Stamp

PIPEA_FRMTIMESTAMP—Pipe A Frame Time Stamp	
Register Type:	MMIO
Address Offset:	**70048h-7004Bh**
Project:	All
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32
Bit	**Description**
31:0	**Pipe_Frame_Time_Stamp** Project: All Format: Provides read back of the display pipe frame time stamp. The time stamp value is sampled at every start of vertical blank. The TIMESTAMP register has information on the time stamp value.

4.2.4 PIPEA_FLIPTIMESTAMP—Pipe A Flip Time Stamp

PIPEA_FLIPTIMESTAMP—Pipe A Flip Time Stamp	
Register Type:	MMIO
Address Offset:	**7004Ch-7004Fh**
Project:	All
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32
Bit	**Description**
31:0	**Pipe_Flip_Time_Stamp** Project: All Format: Provides read back of the display pipe flip time stamp. The time stamp value is sampled on each flip of the surface of the primary plane on this pipe. This includes command streamer asynchronous and synchronous flips and any MMIO writes to the primary plane surface address. The TIMESTAMP register has information on the time stamp value.

4.3 Display Timestamp

This register provides an elapsed real-time value that can be used as a timestamp for GPU events over short periods of time.

The register counts at a constant frequency by adjusting the increment amount according to the actual core display clock frequency. SW therefore does not need to know the reference clock frequency.

4.3.1 TIMESTAMP_HI—Time Stamp High Value

TIMESTAMP_HI—Time Stamp High Value	
Register Type:	MMIO
Address Offset:	**70070h-70073h**
Project:	All
Default Value:	00000000h
Access:	R/W Clear
Size (in bits):	32

Bit	Description
31:0	**TIMESTAMP_High** Project: All Format: This field increments every microsecond. The value in this field is latched in the Pipe Flip TIMESTAMP registers when flips occur, and in the Pipe Frame TIMESTAMP registers at start of vertical blank. The register value will reset if any value is written to it. The register is not reset by a graphics software reset.

4.4 Display Pipeline B

4.4.1 PIPEB_DSL—Pipe B Display Scan Line

PIPEB_DSL—Pipe B Display Scan Line	
Register Type:	MMIO
Address Offset:	**71000h-71003h**
Project:	All
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32

See Pipe A description

Bit	Description
31	**Current_Field** Project: All Default Value: 0b Provides read back of the current field being displayed on display pipe B.
30:13	**Reserved** Project: All Format: MBZ

Value	Name	Description	Project
0b	First	First field (odd field)	All
1b	Second	Second field (even field)	All

PIPEB_DSL—Pipe B Display Scan Line	
12:0	**Line_Counter_for_Display** Project: All Format: See pipe A description.

4.4.2 PIPEB_SLC—Pipe B Display Scan Line Count Range Compare

PIPEB_SLC—Pipe B Display Scan Line Count Range Compare	
Register Type:	MMIO
Address Offset:	**71004h-71007h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
See Pipe A description	
Bit	**Description**
31:13	**Reserved** Project: **DevSNB** Format: MBZ
12:0	**Scan_Line_Number** Project: **Pre-DevSNB:D2** Range 0..Vertical Total See pipe A description
12:0	**Reserved** Project: DevSNB:D2+ Format: MBZ

4.4.3 PIPEBCONF—Pipe B Configuration Register

PIPEBCONF—Pipe B Configuration Register	
Register Type:	MMIO
Address Offset:	**71008h-7100Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank OR pipe disabled

Bit	Description
31	**Pipe_B_Enable** Project: All Default Value: 0b Setting this bit to the value of one, turns on pipe B. This must be done before any planes are enabled on this pipe. Changing it to a zero should only be done when all planes that are assigned to this pipe have been disabled. Turning the pipe enable bit off disables the timing generator in this pipe. Synchronization pulses to the display are not maintained if the timing generator is disabled. Pipe timing registers must contain valid values before this bit is enabled.

Value	Name	Description	Project
0b	Disable	Disable	All
1b	Enable	Enable	All

Bit	Description
30	**Pipe_State** Project: All Default Value: 0b This bit indicates the actual state of the pipe. Since there can be some delay between disabling the pipe and the pipe actually shutting off, this bit indicates the true current state of the pipe.

Value	Name	Description	Project
0b	Disable	Disable	All
1b	Enable	Enable	All

Bit	Description
29	**Reserved** Project: All Format:
28:27	**Reserved**
26	**Reserved** Project: DevSNB Format:

PIPEBCONF—Pipe B Configuration Register

25:24

Pipe_B_Palette/Gamma_Unit_Mode

Project: All

Default Value: 0b

These bits select which mode the pipe gamma correction logic works in. See the Display Palette Registers for information on the different palette/gamma modes. Other gamma units such as in the sprite are unaffected by this bit.

Value	Name	Description	Project
00b	8 bit	8-bit Legacy Palette Mode	All
01b	10 bit	10-bit Precision Palette Mode	All
10b	12 bit	12-bit Interpolated Gamma Mode	All
11b	Reserved	Reserved	All

23:21

Interlaced_Mode

Project: All

Default Value: 0b

These bits are used for software control of interlaced behavior. They are updated immediately if the pipe is off, or in the vertical blank after programming if pipe is enabled.

Note: VGA display modes do not work while in interlaced fetch modes

Value	Name	Description	Project
000b	PF-PD	Progressive Fetch / Progressive display	All
001b	PF-ID	Progressive Fetch / Interlaced display (HDMI) **Requires panel fitting to be enabled**	All
010b	Reserved	Reserved	All
011b	IF-ID	Interlaced Fetch / Interlaced display (programmable sync, normal interlaced)	All
100b	IF-ID-DBL	Interlaced embedded panel with interlaced fetch (pixel doubling power savings mode, no PF enabled)	All
101b	PF-ID-DBL	Interlaced embedded panel with progressive fetch (pixel doubling power savings mode with PF enabled) **Requires panel fitting to be enabled in progressive mode**	All
11Xb	Reserved	Reserved	All

PIPEBCONF—Pipe B Configuration Register

20	**Display_Power_Mode_Switch** Project: All Default Value: 0b This bit is used for software to set the power saving progressive mode. The pipe enters or exits the power savings mode on the vblank after this bit is written. Please note that bits 17:16 of this register must be set to 00 in order for this bit to take effect. If LVDS clockgating is disabled (bits 14, 30 set to '1' in 0xC2020), then clockgating must be temporarily enabled (bits 14, 30 cleared to '0') when toggling Display Power Mode Switch followed by wait of 2 vblanks and then disabled again.

Value	Name	Description	Project
0b	Progressive	Pipe is in progressive mode	All
1b	Power Save	Pipe is in power savings progressive mode	All

19:18	**MSA_Timing_Delay** Project: DevSNB Default Value: 00b This field selects the vertical blank line on which MSA is sent. It is intended for use with embedded DisplayPort panels that support sDRRS. The sDRRS timing switch shall occur on same line as the MSA.

Value	Name	Description	Project
00b	Line1	MSA and sDRRS timing switch occur within the first line of vertical blank	All
01b	Line2	MSA and sDRRS timing switch occur within the second line of vertical blank	All
10b	Line3	MSA and sDRRS timing switch occur within the third line of vertical blank	All
11b	Line4	MSA and sDRRS timing switch occur within the fourth line of vertical blank	All

PIPEBCONF—Pipe B Configuration Register

17:16

Refresh_Rate_Power_Savings_Mode_Association

Project: All

Default Value: 00b

These bits select how refresh rates are switched on pipe B. When they are set to anything other than 00, bits 23:21 of this register must be programmed to 000. Switching between 01 and 10 settings directly is not allowed. Software is responsible for enabling this mode only for integrated display panels that support corresponding mode. The refresh rate hardware control register has additional settings for hardware controlled refresh rate switching.

Value	Name	Description	Project
00b	None	No dynamic refresh rate change enabled. Software control through bits 23:21 only	All
01b	PTP	Progressive-to-progressive refresh rate change enabled. For the CPU and PCH, link and data M and N 1 values are used for high power settings, M and N 2 values for low power settings. For the PCH, pixel clock FPB0 values are used for high power settings, FPB1 values for low power settings.	All
10b	PTI	Progressive-to-interlaced refresh rate change enabled. Pixel clock value does not change in this case. However, data and link M values in the CPU and PCH (as appropriate) are divided by 2 (shifted right by 1 bit). If scaling is enabled, the planes will fetch progressive data which will be interlaced by the panel fitter. If scaling is disabled, the planes will fetch interlaced data, reducing the amount of data fetched	All
11b	Reserv ed	Reserved	All

15:14

Display_Rotation_Info

Project: All

Default Value: 0b

These are informative bits set by software to indicate this pipe is being rotated. Software should set these for both hardware and software rotation cases. Hardware rotation is **not** enabled through these bits.

Value	Name	Description	Project
00b	None	No rotation on this pipe	All
01b	90	90° rotation on this pipe	All
10b	180	180° rotation on this pipe	All
11b	270	270° rotation on this pipe	All

PIPEBCONF—Pipe B Configuration Register

13 — **Color_Range_Select**

Project: All

Default Value: 0b

This bit is used to select the color range of outputs.

Value	Name	Description	Project
0b	Full	Apply full 0-2^n - 1 color range to the output	All
1b	CE	Apply CE color range to the output	All

12:11 — **Pipe_output_color_space_select**

Project: All

Default Value: 0b

Informs the ports of the pipe output color space. Plane data formats and CSC need to be programmed to match what is selected here.

Value	Name	Description	Project
00b	RGB	RGB	All
01b	YUV 601	YUV 601	All
10b	YUV 709	YUV 709	All
11b	Reserved	Reserved	All

10:9 — **Reserved** Project: DevSNB

8 — **Reserved** Project: All Format: MBZ

7:5 — **Bits_Per_Color**

Project: All

Default Value: 0b

This field selects the number of bits per color sent to a receiver device connected to this pipe. Color format takes place on the Vblank after being written. Color format change can be done independent of a pixel clock change.

Software should enable dithering in the pipe/port if selecting a pixel color depth higher or lower than the pixel color depth of the frame buffer.

For further details on Display Port fixed frequency programming to accommodate these formats refer to “DP Frequency Programming” in DPLL section of Bspec.

Value	Name	Description	Project
000b	8 bits	8 bits per color	All
001b	10 bits	10 bits per color	All
010b	6 bits	6 bits per color	All
011b	12 bits	12 bits per color	All
1XXb	Reserved	Reserved	All

PIPEBCONF—Pipe B Configuration Register	
4	**Dithering_enable** Project: All Default Value: 0b This bit enables dithering

Value	Name	Description	Project
0b	Disable	Dithering disabled	All
1b	Enable	Dithering enabled	All

3:2	**Dithering_type** Project: All Default Value: 0b Security: Test These bits select dithering type.

Value	Name	Description	Project
00b	Spatial	Spatial only	All
01b	ST1	Spatio-Temporal 1	All

1	**Reserved**
0	**Reserved** Project: All Format: MBZ

4.5 Display Pipeline B Counters and Timestamps

4.5.1 PIPEB_FRMCOUNT—Pipe B Frame Counter

PIPEB_FRMCOUNT—Pipe B Frame Counter	
Register Type:	MMIO
Address Offset:	**71040h-71043h**
Project:	All
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32
Bit	**Description**
31:0	**Pipe_Frame_Counter** Project: All Format: See Pipe A description

4.5.2 PIPEB_FLIPCOUNT—Pipe B Flip Counter

PIPEB_FLIPCOUNT—Pipe B Flip Counter	
Register Type:	MMIO
Address Offset:	**71044h-71047h**
Project:	All
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32

Bit	Description
31:0	**Pipe_Flip_Counter** Project: All Format: See Pipe A description

4.5.3 PIPEB_FRMTIMESTAMP—Pipe B Frame Time Stamp

PIPEB_FRMTIMESTAMP—Pipe B Frame Time Stamp	
Register Type:	MMIO
Address Offset:	**71048h-7104Bh**
Project:	All
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32

Bit	Description
31:0	**Pipe_Frame_Time_Stamp** Project: All Format: See Pipe A description

4.5.4 PIPEB_FLIPTIMESTAMP—Pipe B Flip Time Stamp

PIPEB_FLIPTIMESTAMP—Pipe B Flip Time Stamp	
Register Type:	MMIO
Address Offset:	**7104Ch-7104Fh**
Project:	All
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32

Bit	Description
31:0	**Pipe_Flip_Time_Stamp** Project: All Format: See Pipe A description

4.6 Cursor A Plane Control Registers

The CURACNTR active register will be updated on the vertical blank or when pipe is disabled, after the CURABASE or CURAPOPUPBASE trigger register is written, or when cursor A is not yet enabled – thus providing an atomic update of the cursor A control and base address registers.

4.6.1 CURACNTR—Cursor A Control Register

CURACNTR—Cursor A Control Register

Register Type:	MMIO
Address Offset:	**70080h-70083h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled or cursor disabled, after armed
Double Buffer Armed By:	Write to CURABASE or CURAVGAPOPUPBASE

For Hi-res modes Cursor A is connected to pipe A only. For VGA popup it follows the VGA pipe select.

Bit	Description
31:28	**Reserved** Project: All Format:
27	**Popup_Cursor_Enabled** Project: All Default Value: 0b This bit should be turned on when using Cursor A as a popup cursor. When in popup mode, hardware interprets the cursor base address as a physical address instead of a graphics address.
26	**Cursor_Gamma_Enabled** Project: All Default Value: 0b This bit only has an effect when using the cursor in a non-VGA mode. In VGA pop-up operation, the cursor data will always bypass the gamma (palette) unit.
25	**Reserved** Project: All Format:

Value	Name	Description	Project
0b	Hi-Res	Cursor A is hi-res	All
1b	VGA	Cursor A is popup	All

Value	Name	Description	Project
0b	Bypass	Cursor pixel data bypasses gamma correction or palette	All
1b	Gamma	Cursor pixel data is gamma to be corrected in the pipe	All

CURACNTR—Cursor A Control Register

24 — **Pipe_Color_Space_Conversion_Enable**

Project: All

Default Value: 0b

This bit enables pipe color space conversion for the cursor pixel data. CSC mode in the pipe CSC registers must be set to match the format of the cursor pixel data.

Value	Name	Description	Project
0b	Bypass	Cursor pixel data bypasses the pipe color space conversion logic	All
1b	Pass	Cursor pixel data passes through the pipe color space conversion logic	All

23:16 — **Reserved** Project: All Format:

15 — **180°_Rotation**

Project: All

Default Value: 0b

This mode causes the cursor to be rotated 180°. Only 32 bits per pixel cursors can be rotated. This field must be zero when the cursor format is 2 bits per pixel.

Value	Name	Description	Project
0b	None	No rotation	All
1b	180	180° Rotation of 32 bit per pixel cursors	All

14 — **Trickle_Feed_Enable**

Project: DevSNB

Default Value: 0b

Value	Name	Description	Project
0b	Enable	Trickle Feed Enabled - Data requests are sent whenever there is space in the Display Data Buffer	All
1b	Disable	Trickle Feed Disabled - Data requests are sent in bursts	All

13:6 — **Reserved** Project: All Format:

5 — **Cursor_Mode_Select[5]**

Project: All

Default Value: 0b

Cursor Mode Select

Bit 5	Bits 2:0	Mode
0	000	Cursor is disabled. This is the default after reset. When the cursor register value changes from enabled to disabled, the cursor will stop fetching data at the following VBLANK event. The cursor enable can be overridden by the pipe cursor disable bit. The value of these bits do not change when disabled by the pipe cursor disable bit.
0	001	Reserved

	0	010	128 x 128 32bpp AND/INVERT (not available for VGA use) See description off 64 x 64 32bpp AND/INVERT format for byte usage
	0	011	256 x 256 32bpp AND/INVERT (not available for VGA use) See description off 64 x 64 32bpp AND/INVERT format for byte usage
	0	100	64 x 64 2bpp Indexed 3-color and transparency mode
	0	101	64 x 64 2bpp Indexed AND/XOR 2-plane mode
	0	110	64 x 64 2bpp Indexed 4-color mode
	0	111	64 x 64 32bpp AND/INVERT (not available for VGA use) For each pixel: Least significant three bytes provides cursor RGB 888 color information Most Significant Byte: All Ones: Opaque, show the cursor color All Zeros: Transparent (color must also equal zero) Other: Invert the underlying display pixel data (ignore the color)
	1	000	Reserved
	1	001	Reserved
	1	010	128 x 128 32bpp ARGB (8:8:8:8 MSB-A:R:G:B) (not available for VGA use)
	1	011	256 x 256 32bpp ARGB (8:8:8:8 MSB-A:R:G:B) (not available for VGA use)
	1	100	64 x 64 32bpp AND/XOR (not available for VGA use) For each pixel: Least significant three bytes provides cursor RGB 888 color information Most Significant Byte: All Ones: Opaque, show the cursor color All Zeros: Transparent (color must also equal zero) Other: XOR the cursor color with the underlying display pixel data
	1	101	128 x 128 32bpp AND/XOR (not available for VGA use) See description off 64 x 64 32bpp AND/XOR format for byte usage
	1	110	256 x 256 32bpp AND/XOR (not available for VGA use) See description off 64 x 64 32bpp AND/XOR format for byte usage
	1	111	64 x 64 32bpp ARGB (8:8:8:8 MSB-A:R:G:B) (not available for VGA use)

4:3	**Reserved** Project: All Format:
2:0	**Cursor_Mode_Select[2:0]** Project: All Default Value: 0b These three bits together with bit 5 select the mode for cursor as shown in the table described in bit 5.

4.6.2 CURABASE—Cursor A Base Address Register

CURABASE—Cursor A Base Address Register	
Register Type:	MMIO
Address Offset:	**70084h-70087h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

Writes to this register arm CURA registers

This register is only used when cursor A is in the hi-res mode. In VGA popup mode CURAVGAPOPUPBASE is used instead and this register must not be written. This register specifies the graphics memory address at which the cursor image data is located.

Bit	Description
31:12	**Cursor_Base_Address[31:12]** Project: All Address: GraphicsAddress[31:12] This field specifies bits 31:12 of the graphics address of the base of the cursor for hi-res mode. The physical address used for VGA popup cursor is in the CURAVGAPOPUPBASE register. The cursor surface address must be 4K byte aligned. The cursor must be in linear memory, it cannot be tiled. When performing 180° rotation, this address does not change, hardware will internally offset to start from the last pixel of the last line of the cursor.
11:0	**Reserved** Project: All Format:

4.6.3 CURAPOS—Cursor A Position Register

CURAPOS—Cursor A Position Register	
Register Type:	MMIO
Address Offset:	**70088h-7008Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

This register specifies the screen position of the cursor. The origin of the cursor position is always the upper left corner of the active image for the display pipe that the cursor is assigned.

Bit	Description
31	**Cursor_Y-Position_Sign_Bit** Project: All This bit provides the sign bit of a signed 13-bit value that specifies the horizontal position of cursor. For normal high resolution display modes, the cursor must have at least a single pixel positioned over the active screen. For use as a VGA Popup, the entire cursor must be positioned over the active area of the VGA image.
30:28	**Reserved** Project: All Format: MBZ
27:16	**Cursor_Y-Position_Magnitude_Bits** Project: All This register provides the magnitude bits of a signed 12-bit value that specifies the vertical position of cursor. The sign bit of this value is provided by bit 31of this register. For use as a VGA Popup, the entire cursor must be positioned over the active area of the VGA image. Enabling the border in VGA (VGA Border Enable bit in the VGA Config register) includes the border in what is considered the "active area". When performing 180° rotation, this field specifies the vertical position of the lower right corner relative to the end of the active video area in the unrotated orientation
15	**Cursor_X-Position_Sign_Bit** Project: All This bit provides the sign bit of a signed 13-bit value that specifies the horizontal position of cursor. For normal high resolution display modes, the cursor must have at least a single pixel positioned over the active screen. For use as a VGA Popup, the entire cursor must be positioned over the active area of the VGA image. Enabling the border in VGA (VGA Border Enable bit in the VGA Config register) includes the border in what is considered the "active area".
14:12	**Reserved** Project: All Format: MBZ
11:0	**Cursor_X-Position_Magnitude_Bits** Project: All These 12 bits provide the signed 13-bit value that specifies the horizontal position of cursor. The sign bit is provided by bit 15 of this register. When performing 180° rotation, this field specifies the horizontal position of the lower right corner relative to the end of the active video area in the unrotated orientation.

4.6.4 CURAVGAPOPUPBASE—Cursor A VGA Popup Base Address Register

CURAVGAPOPUPBASE—Cursor A VGA Popup Base Address Register	
Register Type:	MMIO
Address Offset:	**7008Ch-7008Fh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

Writes to this register arm CURA registers

This register is only used when cursor A is in the VGA popup mode. In hi-res mode CURABASE is used instead and this register must not be written. This register specifies the physical memory address at which the cursor image data is located.

Bit	Description
31:12	**Cursor_VGA_Popup_Base_Address[31:12]** Project: All Address: PhysicalAddress[31:12] This field specifies bits 31:12 of the physical address of the base of the cursor for VGA popup mode. The graphics address used for hi-res cursor is in the CURABASE register. The cursor surface address must be 4K byte aligned. The cursor must be in linear memory, it cannot be tiled.
11:7	**Reserved** Project: All Format:
6:0	**Cursor_VGA_Popup_Base_Address_MSBs[38:32]** Project: All Address: PhysicalAddress[38:32] This field specifies bits 38:32 of the physical address of the base of the cursor for VGA popup mode. See restrictions in Cursor VGA Popup Base Address field.

4.6.5 CURAPALET—Cursor A Palette registers

Cursor Palette Format						
Project:	All					
Bit	**Description**					
31:24	**Reserved**	Project:	All		Format:	MBZ
23:16	**Red_or_Y_Value**	Project:	All	Format:		
	These registers specify the cursor palette. RGB data is full range unsigned numbers. YUV data will be unsigned for the Y and excess 128 notation for the UV values. The data can be pre-gamma corrected and bypass the gamma correction logic or passed through the gamma corrector.					
15:8	**Green_or_U_Value**	Project:	All	Format:		
7:0	**Blue_or_V_Value**	Project:	All	Format:		

CURAPALET—Cursor A Palette registers

Register Type:	MMIO
Address Offset:	**70090h-7009Fh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	4x32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

The cursor palette provides color information when using one of the indexed modes. The two-bit index selects one of the four colors or two of the colors when in the AND/XOR cursor mode. The cursor palette provides color information when using one of the indexed modes. The two-bit index selects one of the four colors or two of the colors when in the AND/XOR cursor mode.

The table below describes the palette usage for different cursor modes and indexes.

Index	2 color	3color	4color
00	palette 0	palette 0	palette 0
01	palette 1	palette 1	palette 1
10	transparent	transparent	palette 2
11	invert destination (palette 3 all 1s)	palette 3	palette 3

Palette 3 must be programmed with all 1s for invert destination.

DWord	Bit	Description				
0	31:0	**CURAPALET0**	Project:	All	Format:	Cursor Palette Format
1	31:0	**CURAPALET1**	Project:	All	Format:	Cursor Palette Format

CURAPALET—Cursor A Palette registers						
2	31:0	**CURAPALET2**	Project:	All	Format:	Cursor Palette Format
3	31:0	**CURAPALET3**	Project:	All	Format:	Cursor Palette Format

4.6.6 CURASURFLIVE—Cursor A Live Surface Base Address

CURASURFLIVE—Cursor A Live Surface Base Address	
Register Type:	MMIO
Address Offset:	**700ACh-700AFh**
Project:	All
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32

Bit	Description
31:0	**Cursor_A_Live_Surface_Base_Address** Project: All Format: This gives the live value of the surface base address as being currently used for the plane.

4.7 Cursor B Plane Control Registers

The CURBCNTR active register will be updated on the vertical blank or when pipe is disabled, after the CURBBASE trigger register is written, or when cursor B is not yet enabled – thus providing an atomic update of the cursor B control and base address registers.

4.7.1 CURBCNTR—Cursor B Control Register

CURBCNTR—Cursor B Control Register

Register Type:	MMIO
Address Offset:	**700C0h-700C3h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled or cursor disabled, after armed
Double Buffer Armed By:	Write to CURBBASE

Cursor B is connected to pipe B only.

Bit	Description
31:27	**Reserved** Project: All Format: MBZ
26	**Cursor_Gamma_Enable** Project: All Default Value: 0b

Value	Name	Description	Project
0b	Bypass	Cursor pixel data bypasses gamma correction	All
1b	Corrected	Cursor pixel data is gamma to be corrected	All

25	**Reserved** Project: All Format:
24	**Pipe_Color_Space_Conversion_Enable** Project: All Default Value: 0b This bit enables pipe color space conversion for the cursor pixel data. CSC mode in the pipe CSC registers must be set to match the format of the cursor pixel data

Value	Name	Description	Project
0b	Bypass	Cursor pixel data bypasses the pipe color space conversion logic	All
1b	pass	Cursor pixel data passes through the pipe color space conversion logic.	All

23:16	**Reserved** Project: All Format:
15	**180°_Rotation** Project: All Default Value: 0b This mode causes the cursor to be rotated 180°. Only 32 bits per pixel cursors can be rotated. This field must be zero when the cursor format is 2 bits per pixel.

Value	Name	Description	Project
0b	None	No rotation	All
1b	180	180° Rotation of 32 bit per pixel cursors	All

CURBCNTR—Cursor B Control Register	
14	**Trickle_Feed_Enable** Project: DevSNB Default Value: 0b

Value	Name	Description	Project
0b	Enable	Trickle Feed Enabled - Data requests are sent whenever there is space in the Display Data Buffer.	All
1b	Disable	Trickle Feed Disabled - Data requests are sent in bursts.	All

13:6	**Reserved** Project: All Format: MBZ
5	**Cursor_Mode_Select** Project: All Format: **Cursor Mode Select** Defined in CURACNTR—Cursor A Control Register Bit 5.
4:3	**Reserved** Project: All Format:
2:0	**Cursor_Mode_Select** Project: All Format: These three bits together with bit 5 select the mode for cursor as shown in CURACNTR—Cursor A Control Register Bit 5.

4.7.2 CURBBASE—Cursor B Base Address Register

CURBBASE—Cursor B Base Address Register	
Register Type:	MMIO
Address Offset:	**700C4h-700C7h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

Writes to this register arm CURB registers

This register specifies the graphics memory address at which the cursor image data is located.

Bit	Description
31:12	**Cursor_Base_Address[31:12]** Project: All Address: GraphicsAddress[31:12] This register specifies the graphics address of the cursor. It also acts as a trigger event to force the update of active registers on the next display event. The cursor surface address must be 4K byte aligned. The cursor must be in linear memory, it cannot be tiled. When performing 180° rotation, this address does not change, hardware will internally offset to start from the last pixel of the last line of the cursor.
11:0	**Reserved** Project: All Format: MBZ

4.7.3 CURBPOS—Cursor B Position Register

CURBPOS—Cursor B Position Register	
Register Type:	MMIO
Address Offset:	**700C8h-700CBh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

This register specifies the screen position of the cursor. The origin of the cursor position is always the upper left corner of the active image for the display pipe that the cursor is assigned.

Bit	Description
31	**Cursor_Y-Position_Sign_Bit** Project: All This bit provides the sign bit of a signed 13-bit value that specifies the horizontal position of cursor. For normal high resolution display modes, the cursor must have at least a single pixel positioned over the active screen.
30:28	**Reserved** Project: All Format: MBZ
27:16	**Cursor_Y-Position_Magnitude_Bits** Project: All This register provides the magnitude bits of a signed 13-bit value that specifies the vertical position of cursor. The sign bit of this value is provided by bit 31of this register. When performing 180° rotation, this field specifies the vertical position of the lower right corner relative to the end of the active video area in the unrotated orientation.
15	**Cursor_X-Position_Sign_Bit** Project: All This bit provides the sign bit of a signed 13-bit value that specifies the horizontal position of cursor. For normal high resolution display modes, the cursor must have at least a single pixel positioned over the active screen.
14:12	**Reserved** Project: All Format: MBZ
11:0	**Cursor_X-Position_Magnitude_Bits** Project: All These 12 bits provide the signed 13-bit value that specifies the horizontal position of cursor. The sign bit is provided by bit 15 of this register. When performing 180° rotation, this field specifies the vertical position of the lower right corner relative to the end of the active video area in the unrotated orientation.

4.7.4 CURBPALET—Cursor B Palette registers

CURBPALET—Cursor B Palette registers	
Register Type:	MMIO
Address Offset:	**700D0h-700DFh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	4x32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

The cursor palette provides color information when using one of the indexed modes. The two-bit index selects one of the four colors or two of the colors when in the AND/XOR cursor mode. The cursor palette provides color information when using one of the indexed modes. The two-bit index selects one of the four colors or two of the colors when in the AND/XOR cursor mode. See Cursor A palette usage table.

DWord	Bit	Description
0	31:0	**CURBPALET0** Project: All Format: Cursor Palette Format
1	31:0	**CURBPALET1** Project: All Format: Cursor Palette Format
2	31:0	**CURBPALET2** Project: All Format: Cursor Palette Format
3	31:0	**CURBPALET3** Project: All Format: Cursor Palette Format

4.7.5 CURBSURFLIVE—Cursor B Live Surface Base Address Register

CURBSURFLIVE—Cursor B Live Surface Base Address Register	
Register Type:	MMIO
Address Offset:	**700ECh-700Efh**
Project:	All
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32

Bit	Description
31:0	**Cursor_Live_Surface_Base_Address** Project: All Format: This gives the live value of the surface base address as being currently used for the plane.

4.8 Primary A Plane Control

The DSPACNTR and DSPASTRIDE active registers will be updated on the vertical blank or when pipe is disabled, after the DSPASURF trigger register is written, or when the primary A is not yet enabled – thus providing an atomic update of the primary A control, stride, and base address registers.

4.8.1.1 DSPACNTR—Primary A Control Register

DSPACNTR—Primary A Control Register

Register Type:	MMIO
Address Offset:	**70180h-70183h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled or primary disabled, after armed
Double Buffer Armed By:	Write to DSPASURF

Primary A Plane is connected to pipe A only.

Bit	Description
31	**Primary_Plane_Enable** Project: All Format: Enable When this bit is set, the primary plane will generate pixels for display. When set to zero, primary plane memory fetches cease and plane output is transparent. The display pipe must be enabled to enable the plane. When in Self Refresh Big FIFO mode, write to this register to enable the plane will be internally buffered and delayed while Big FIFO mode is exiting.
30	**Gamma_Enable** Project: All Default Value: 0b This bit should only be changed after the plane has been disabled. It controls the bypassing of the display pipe gamma unit for the plane pixel data. For 8-bit indexed display data, this bit should be set to a one.

Value	Name	Description	Project
0b	Bypass	Plane pixel data bypasses the display pipe gamma correction logic	All
1b	Correct	Plane pixel data is gamma corrected in the display pipe gamma correction logic.	All

DSPACNTR—Primary A Control Register

29:26

Source_Pixel_Format

Project: All

Default Value: 0b

These bits should only be changed after the plane has been disabled. Pixel format of 8-bit indexed uses the pipe palette. Before entering the blender, each source format is converted to 12 bits per pixel.

Value	Name	Description	Project
0010b	8bpp	8-bpp Indexed	All
0101b	16-bit BGRX (5:6:5 MSB-R:G:B)	pixel format (XGA compatible).	All
0110b	32-bit BGRX (8:8:8:8 MSB-X:R:G:B)	pixel format. Ignore alpha	All
1000b	32-bit RGBX (2:10:10:10 MSB-X:B:G:R)	pixel format. Ignore alpha	All
1010b	32-bit BGRX (2:10:10:10 MSB-X:R:G:B)	pixel format. Ignore alpha	All
1100b	64-bit RGBX Floating Point(16:16:16:16 MSB-X:B:G:R) Use of 64bpp format will limit the maximum dot clock to 80% of cdclk	pixel format. Ignore alpha	All
1110b	32-bit RGBX (8:8:8:8 MSB-X:B:G:R)	pixel format. Ignore alpha	All
others	Reserved	Reserved	All

25

Reserved Project: DevSNB Format:

24

Pipe_Color_Space_Conversion_Enable

Project: All

Default Value: 0b

This bit enables pipe color space conversion for the plane pixel data. CSC mode in the pipe CSC registers must be set to match the format of the plane pixel data.

Value	Name	Description	Project
0b	Bypass	Plane pixel data bypasses the pipe color space conversion logic	All
1b	Pass	Plane pixel data passes through the pipe color space conversion logic.	All

23:16

Reserved Project: All Format:

15

180°_Display_Rotation

Project: All

Default Value: 0b

This mode causes the plane to be rotated 180°. In addition to setting this bit, software must also set the surface address offset to the lower right corner of the unrotated image.

Value	Name	Description	Project
0b	None	No rotation	All
1b	180	180° rotation	All

DSPACNTR—Primary A Control Register

14 — **Trickle_Feed_Enable**

Project: All

Default Value: 0b

This bit must always be programmed to '1'.

Value	Name	Description	Project
0b	Enable	Trickle Feed Enabled - Plane data requests are sent whenever there is space in the Display Data Buffer.	All
1b	Disable	Trickle Feed Disabled - Plane data requests are sent in bursts	All

12:11 — **Reserved** Project: All Format:

10 — **Tiled_Surface**

Project: All

Default Value: 0b

This bit indicates that the plane surface data is in tiled memory. Only X tiling is supported for display surfaces.

When this bit is set, it affects the hardware interpretation of the DSPATILEOFF, DSPALINOFF, and DSPASURF registers.

Value	Name	Description	Project
0b	Linear	Plane uses linear memory	All
1b	X-tiled	Plane uses X-tiled memory	All

9 — **Asynchronous_Surface_Address_Update_Enable**

Project: All

Default Value: 0b

This bit will enable asynchronous updates of the surface address when written by MMIO. The surface address will change with the next TLB request or when start of vertical blank is reached. Updates during vertical blank may not complete until after the first few active lines are displayed.

Restrictions:

- No command streamer initiated surface address updates to this plane are allowed when this bit is enabled.
- Wait for flip done indication in pipe status register before writing the surface address register again with this bit set.

Value	Name	Description	Project
0b	Sync	Surface Address MMIO writes will update synchronous to start of vertical blank	All
1b	Async	Surface Address MMIO writes will update asynchronously	All

8:0 — **Reserved** Project: All Format: MBZ

Primary Plane Source Pixel Format Mapping of Bits to Colors:

Note: For 64-bit Floating Point format, each of the 16 bit color components is 1:5:10 MSB-sign:exponent:fraction

PRIMARY RGB	Ignored	Red	Green	Blue
16-bit BGRX 5:6:5	N/A	15:11	10:5	4:0
32-bit BGRX 8:8:8	31:24	23:16	15:8	7:0
32-bit RGBX 10:10:10	31:30	9:0	19:10	29:20
32-bit BGRX 10:10:10	31:30	29:20	19:10	9:0
64-bit RGBX Float 16:16:16	63:48	15:0	31:16	47:32
32-bit RGBX 8:8:8	31:24	7:0	15:8	23:16

4.8.2 DSPALINOFF—Primary A Linear Offset Register

DSPALINOFF—Primary A Linear Offset Register

Register Type:	MMIO
Address Offset:	**70184h-70187h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

Bit	Description
31:0	**Primary_Linear_Offset** Project: All Format: This register provides the linear panning byte offset into the primary plane. This value is added to the surface address to get the address of the first pixel to be displayed. This offset must be at least pixel aligned. When performing 180° rotation, the unpanned offset must be the difference between the last pixel of the last line of the display data in its unrotated orientation and the display surface address. When the surface is tiled, the contents of this register are ignored.

4.8.3 DSPASTRIDE—Primary A Stride Register

DSPASTRIDE—Primary A Stride Register	
Register Type:	MMIO
Address Offset:	**70188h-7018Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled or primary disabled, after armed
Double Buffer Armed By:	Write to DSPASURF

Bit	Description
31:16	**Reserved** Project: All Format:
15:6	**Primary_Stride** Project: All Format: This is the stride for the primary plane in bytes. When using linear memory, this must be 64 byte aligned. When using tiled memory, this must be 512 byte aligned. This value is used to determine the line to line increment for the display. This register is updated through either a command packet passed through the command stream or writes to this register. The stride is limited to a maximum of 32K bytes for both linear and tiled memory.
5:0	**Reserved** Project: All Format:

4.8.4 DSPASURF—Primary A Surface Base Address Register

DSPASURF—Primary A Surface Base Address Register	
Register Type:	MMIO
Address Offset:	**7019Ch-7019Fh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

Writes to this register arm DSPA registers

DSPASURF—Primary A Surface Base Address Register	
Bit	**Description**
31:12	**Primary_Surface_Base_Address** Project: All Address: GraphicsAddress[31:12] This address specifies the surface base address. When the surface is tiled, panning is specified using (x, y) offsets in the DSPATILEOFF register. When the surface is in linear memory, panning is specified using a linear offset in the DSPALINOFF register. This address must be 4K aligned. When performing asynchronous flips and the display surface is in tiled memory, this address must be 256K aligned. This register can be written directly through software or by command packets in the command stream. It represents an offset from the graphics memory aperture base and is mapped to physical pages through the global GTT.
11:3	**Reserved** Project: All Format:
2	**Reserved**
0	**Reserved** Project: All Format: MBZ

4.8.5 DSPATILEOFF—Primary A Tiled Offset Register

DSPATILEOFF—Primary A Tiled Offset Register	
Register Type:	MMIO
Address Offset:	**701A4h-701A7h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

This register specifies the panning for the display surface. The surface base address is specified in the DSPASURF register, and this register is used to describe an offset from that base address. When the surface is in linear memory, the offset is specified in the DSPALINOFF register and the contents of this register are ignored. When the surface is tiled, the start position is specified in this register as an (x, y) offset from the beginning of the surface.

Bit	Description
31:28	**Reserved** Project: All Format: MBZ
27:16	**Primary_Start_Y-Position** Project: All Format: These 12 bits specify the vertical position in lines of the beginning of the active display plane relative to the display surface. When performing 180° rotation, this field specifies the vertical position of the lower right corner relative to the start of the active display plane in the unrotated orientation.
15:12	**Reserved** Project: All Format: MBZ

DSPATILEOFF—Primary A Tiled Offset Register	
11:0	**Primary_Start_X-Position** Project: All Format: These 12 bits specify the horizontal offset in pixels of the beginning of the active display plane relative to the display surface. When performing 180° rotation, this field specifies the horizontal position of the lower right corner relative to the start of the active display plane in the unrotated orientation.

4.8.6 DSPASURFLIVE—Primary A Live Surface Base Address

DSPASURFLIVE—Primary A Live Surface Base Address	
Register Type:	MMIO
Address Offset:	**701ACh-701AFh**
Project:	All
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32
Trusted Type:	1
Bit	**Description**
31:0	**Primary_Live_Surface_Base_Address** Project: All Address: GraphicsAddress[31:0] This gives the live value of the surface base address as being currently used for the plane.

4.9 Primary B Plane Control

The DSPBCNTR and DSPBSTRIDE active registers will be updated on the vertical blank or when pipe is disabled, after the DSPBSURF trigger register is written, or when the primary B is not yet enabled – thus providing an atomic update of the primary B control, stride, and base address registers.

4.9.1 DSPBCNTR—Primary B Control Register

DSPBCNTR—Primary B Control Register

Register Type:	MMIO
Address Offset:	**71180h-71183h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled or primary disabled, after armed
Double Buffer Armed By:	Write to DSPBSURF

Primary B Plane is connected to pipe B only

Bit	Description
31	**Primary_Plane_Enable** Project: All Format: Enable When this bit is set, the primary plane will generate pixels for display. When set to zero, primary plane memory fetches cease and plane output is transparent. The display pipe must be enabled to enable the plane. When in Self Refresh Big FIFO mode, write to this register to enable the plane will be internally buffered and delayed while Big FIFO mode is exiting.
30	**Gamma_Enable** Project: All Default Value: 0b This bit should only be changed after the plane has been disabled. It controls the bypassing of the display pipe gamma unit for the plane pixel data. For 8-bit indexed display data, this bit should be set to a one.

Value	Name	Description	Project
0b	Bypass	Plane pixel data bypasses the display pipe gamma correction logic	All
1b	Correct	Plane pixel data is gamma corrected in the display pipe gamma correction logic.	All

DSPBCNTR—Primary B Control Register

29:26

Source_Pixel_Format

Project: All

Default Value: 0b

These bits should only be changed after the plane has been disabled. Pixel format of 8-bit indexed uses the pipe palette. Before entering the blender, each source format is converted to 12 bits per pixel.

Value	Name	Description	Project
0010b	8bpp	8-bpp Indexed	All
0101b	16-bit BGRX (5:6:5 MSB-R:G:B)	pixel format (XGA compatible).	All
0110b	32-bit BGRX (8:8:8:8 MSB-X:R:G:B)	pixel format. Ignore alpha	All
1000b	32-bit RGBX (2:10:10:10 MSB-X:B:G:R)	pixel format. Ignore alpha	All
1010b	32-bit BGRX (2:10:10:10 MSB-X:R:G:B)	pixel format. Ignore alpha	All
1100b	64-bit RGBX Floating Point(16:16:16:16 MSB-X:B:G:R) Use of 64bpp format will limit the maximum dot clock to 80% of cdclk.	pixel format. Ignore alpha	All
1110b	32-bit RGBX (8:8:8:8 MSB-X:B:G:R)	pixel format. Ignore alpha	All
others	Reserved	Reserved	All

25

Reserved Project: DevSNB

24

Pipe_Color_Space_Conversion_Enable

Project: All

Default Value: 0b

This bit enables pipe color space conversion for the plane pixel data. CSC mode in the pipe CSC registers must be set to match the format of the plane pixel data.

Value	Name	Description	Project
0b	Bypass	Plane pixel data bypasses the pipe color space conversion logic	All
1b	Pass	Plane pixel data passes through the pipe color space conversion logic.	All

23:16

Reserved Project: All Format:

15

180°_Display_Rotation

Project: All

Default Value: 0b

This mode causes the plane to be rotated 180°. In addition to setting this bit, software must also set the surface address offset to the lower right corner of the unrotated image.

Value	Name	Description	Project
0b	None	No rotation	All
1b	180	180° rotation	All

DSPBCNTR—Primary B Control Register

14 — **Trickle_Feed_Enable**

Project: All

Default Value: 0b

This bit must always be programmed to '1'.

Value	Name	Description	Project
0b	Enable	Trickle Feed Enabled - Plane data requests are sent whenever there is space in the Display Data Buffer.	All
1b	Disable	Trickle Feed Disabled - Plane data requests are sent in bursts	All

13 — **Reserved**

12:11 — **Reserved** Project: All Format:

10 — **Tiled_Surface**

Project: All

Default Value: 0b

This bit indicates that the plane surface data is in tiled memory. Only X tiling is supported for display surfaces.

When this bit is set, it affects the hardware interpretation of the DSPBTILEOFF, DSPBLINOFF, and DSPBSURF registers.

Value	Name	Description	Project
0b	Linear	Plane uses linear memory	All
1b	X-tiled	Plane uses X-tiled memory	All

9 — **Asynchronous_Surface_Address_Update_Enable**

Project: All

Default Value: 0b

This bit will enable asynchronous updates of the surface address when written by MMIO. The surface address will change with the next TLB request or when start of vertical blank is reached. Updates during vertical blank may not complete until after the first few active lines are displayed.

Restrictions:

- No command streamer initiated surface address updates to this plane are allowed when this bit is enabled.
- Wait for flip done indication in pipe status register before writing the surface address register again with this bit set.

Value	Name	Description	Project
0b	Sync	Surface Address MMIO writes will update synchronous to start of vertical blank	All
1b	Async	Surface Address MMIO writes will update asynchronously	All

8:0 — **Reserved** Project: All Format: MBZ

See DSPACNTR - Primary Plane Source Pixel Format Mapping of Bits to Colors

4.9.2 DSPBLINOFF—Primary B Linear Offset Register

DSPBLINOFF—Primary B Linear Offset Register	
Register Type:	MMIO
Address Offset:	**71184h-71187h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

Bit	Description
31:0	**Primary_Linear_Offset** Project: All This register provides the linear panning byte offset into the primary plane. This value is added to the surface address to get the address of the first pixel to be displayed. This offset must be at least pixel aligned. When performing 180° rotation, the unpanned offset must be the difference between the last pixel of the last line of the display data in its unrotated orientation and the display surface address. When the surface is tiled, the contents of this register are ignored.

4.9.3 DSPBSTRIDE—Primary B Stride Register

DSPBSTRIDE—Primary B Stride Register	
Register Type:	MMIO
Address Offset:	**71188h-7118Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled or primary disabled, after armed
Double Buffer Armed By:	Write to DSPBSURF

Bit	Description
31:16	**Reserved** Project: All Format: MBZ
15:6	**Primary_Stride** Project: All This is the stride for the primary plane in bytes. When using linear memory, this must be 64 byte aligned. When using tiled memory, this must be 512 byte aligned. This value is used to determine the line to line increment for the display. This register is updated through either a command packet passed through the command stream or writes to this register. The stride is limited to a maximum of 32K bytes for both linear and tiled memory.
5:0	**Reserved** Project: All Format: MBZ

4.9.4 DSPBSURF—Primary B Surface Base Address Register

DSPBSURF—Primary B Surface Base Address Register	
Register Type:	MMIO
Address Offset:	**7119Ch-7119Fh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

Writes to this register arm DSPB registers

Bit	Description
31:12	**Primary_Surface_Base_Address** Project: All Address: GraphicsAddress[31:12] This address specifies the surface base address. When the surface is tiled, panning is specified using (x, y) offsets in the DSPBTILEOFF register. When the surface is in linear memory, panning is specified using a linear offset in the DSPBLINOFF register. This address must be 4K aligned. When performing asynchronous flips and the display surface is in tiled memory, this address must be 256K aligned. This register can be written directly through software or by command packets in the command stream. It represents an offset from the graphics memory aperture base and is mapped to physical pages through the global GTT.
11:3	**Reserved** Project: All Format:
2	**Reserved**
0	**Reserved** Project: All Format: MBZ

4.9.5 DSPBTILEOFF—Primary B Tiled Offset Register

Address Offset: 711A4h

Default: 00000000h

Normal Access: Read/Write
Double Buffer Update Point: Start of vertical blank or pipe disabled

Size: 32 bits

This register specifies the panning for the display surface. The surface base address is specified in the DSPBSURF register, and this register is used to describe an offset from that base address. When the surface is in linear memory, the offset is specified in the DSPBLINOFF register and the contents of this register are ignored. When the surface is tiled, the start position is specified in this register as an (x, y) offset from the beginning of the surface.

Bit	Descriptions
31:28	**Reserved:** Write as zero
27:16	**Primary Start Y-Position:** These 12 bits specify the vertical position in lines of the beginning of the active display plane relative to the display surface. When performing 180° rotation, this field specifies the vertical position of the lower right corner relative to the start of the active display plane in the unrotated orientation.
15:12	**Reserved:** Write as zero
11:0	**Primary Start X-Position:** These 12 bits specify the horizontal offset in pixels of the beginning of the active display plane relative to the display surface. When performing 180° rotation, this field specifies the horizontal position of the lower right corner relative to the start of the active display plane in the unrotated orientation.

4.9.6 DSPBSURFLIVE—Primary B Live Surface Base Address

DSPBSURFLIVE—Primary B Live Surface Base Address	
Register Type:	MMIO
Address Offset:	**711ACh-711AFh**
Project:	All
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32
Bit	**Description**
31:0	**Primary_Live_Surface_Base_Address** Project: All Address: GraphicsAddress[31:0] This gives the live value of the surface base address as being currently used for the plane.

4.10 Video Sprite A Control

Two video sprites are provided for the display of video content. These sprite planes provide windowing and keying functions as well as gamma and color space conversion from YUV to RGB. Each sprite plane is attached to only one of the pipes, Video Sprite A to pipe A and Video Sprite B to pipe B. Apart from the pipe assignments, the functionality is identical.

The DVSACNTR, DVSASTRIDE, DVSAPOS, DVSASIZE, and DVSASCALE active registers will be updated on the vertical blank or when pipe is disabled, after the DVSASURF trigger register is written, or when the sprite A is not yet enabled – thus providing an atomic update of the video sprite A control, stride, position, size, scale, and base address registers.

4.10.1 DVSACNTR—Video Sprite A Control Register

DVSACNTR—Video Sprite A Control Register

Register Type:	MMIO
Address Offset:	**72180h-72183h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled or sprite disabled, after armed
Double Buffer Armed By:	Write to DVSASURF

Video Sprite A Plane is connected to pipe A only.

Bit	Description
31	**Sprite_Enable** Project: All Format: Enable When this bit is set, the sprite plane will generate pixels for display. When set to zero, sprite plane memory fetches cease and plane output is transparent. The display pipe must be enabled to enable the plane. When in Self Refresh Big FIFO mode, write to this register to enable the plane will be internally buffered and delayed while Big FIFO mode is exiting.
30	**Gamma_Enable** Project: All Default Value: 0b There are two gamma adjustments possible in the video sprite data path. This bit controls the gamma correction in the display pipe not the gamma control in this plane. It affects only the pixel data from this display plane. Gamma correction logic that is contained in the video sprite is disabled by loading the default values into those registers.

Value	Name	Description	Project
0b	Disable	Plane pixel data bypasses the display pipe gamma correction logic	All
1b	Enable	Plane pixel data is gamma corrected in the pipe gamma correction logic	All

29	**Reserved** Project: All Format: MBZ
28	**YUV_Bypass_Excess-512_Format_Conversion** Project: All Default Value: 0b

Value	Name	Description	Project
0b	Disable	Disable excess-512 conversion	All
1b	Enable	Enable excess-512 conversion	All

DVSACNTR—Video Sprite A Control Register

27

Range_Correction_Disable

Project: All

Default Value: 0b

Setting this bit disables the YUV range correction logic. Normally the range compressed YUV is expanded to full range RGB, setting this bit will generate range compressed RGB. This bit should also be used if full range YUV source material is used. This bit has no effect on RGB source formats.

Value	Name	Description	Project
0b	Enable	Range correction enabled	All
1b	Disable	No range correction	All

26:25

Source_Pixel_Format

Project: DevSNB

Default Value: 0b

This field selects the pixel format for the sprite. Before entering the blender, each source format is converted to 12 bits per pixel.

Value	Name	Description	Project
00b	YUV 4:2:2	YUV 4:2:2 packed pixel format (byte order programmed separately)	All
01b	RGB 32-bit 2:10:10:10	RGB 32-bit 2:10:10:10 pixel format (color order programmed separately). Ignore alpha.	All
10b	RGB 32-bit 8:8:8:8	RGB 32-bit 8:8:8:8 pixel format (color order programmed separately). Ignore alpha.	All
11b	RGB 64-bit 16:16:16:16 Use of 64bpp format will limit the maximum dot clock to 80% of cdclk.	RGB 64-bit 16:16:16:16 floating point pixel format (color order programmed separately). Ignore alpha.	All

24

Pipe_Color_Space_Conversion_Enable

Project: All

Default Value: 0b

This bit enables pipe color space conversion for the plane pixel data. This is separate from the color conversion logic within the sprite plane. CSC mode in the pipe CSC registers must be set to match the format of the plane pixel data after the color conversion logic within the sprite plane.

Value	Name	Description	Project
0b	Disable	Plane pixel data bypasses the pipe color space conversion logic	All
1b	Enable	Plane pixel data passes through the pipe color space conversion logic	All

23 **Reserved** Project: All Format: MBZ

DVSACNTR—Video Sprite A Control Register

22	**Sprite_Source_Key_Enable** Project: All Default Value: 0b This bit enables source color keying. Sprite pixel values that match (within range) the key will become transparent. Source key can not be enabled if destination key is enabled.

Value	Name	Description	Project
0b	Disable	Sprite source key is disabled	All
1b	Enable	Sprite source key is enabled	All

21	**Reserved** Project: DevSNB Format:
20	**RGB_Color_Order** Project: DevSNB Default Value: 0b This field is used to select the color order when using RGB data formats. For other formats, this field is ignored.

Value	Name	Description	Project
0b	BGRX	BGRX (MSB-X:R:G:B)	All
1b	RGBX	RGBX (MSB-X:B:G:R)	All

20	**Reserved** Project: DevILK Format:
19	**Color_Conversion_Disabled** Project: All Default Value: 0b This bit enables or disables the color conversion logic internal to the sprite. Color conversion is intended to be used with the formats that support YUV format. This bit is ignored when using RGB source formats.

Value	Name	Description	Project
0b	Enable	Pixel data is sent through the sprite color conversion logic (only applies to YUV formats)	All
1b	Disable	Pixel data is not sent through the sprite YUV->RGB color conversion logic.	All

DVSACNTR—Video Sprite A Control Register

18 — **YUV_Format**

Project: All

Default Value: 0b

This bit specifies the source YUV format for the YUV to RGB color conversion operation. This field is ignored when source data is RGB.

Value	Name	Description	Project
0b	BT.601	ITU-R Recommendation BT.601	All
1b	BT.709	ITU-R Recommendation BT.709	All

17:16 — **YUV_Byte_Order**

Project: All

Default Value: 0b

This field is used to select the byte order when using YUV 4:2:2 data formats. For other formats, this field is ignored.

Value	Name	Description	Project
00b	YUYV	YUYV (8:8:8:8 MSB-V:Y_2:U:Y_1)	All
01b	UYVY	UYVY (8:8:8:8 MSB-Y_2:V:Y_1:U)	All
10b	YVYU	YVYU (8:8:8:8 MSB-U:Y_2:V:Y_1)	All
11b	VYUY	VYUY (8:8:8:8 MSB-Y_2:U:Y_1:V)	All

15 — **180°_Display_Rotation**

Project: All

Default Value: 0b

This mode causes the plane to be rotated 180°. In addition to setting this bit, software must also set the surface address offset to the lower right corner of the unrotated and unscaled image and calculate the x, y offset as relative to the lower right corner.

Value	Name	Description	Project
0b	None	No rotation	All
1b	180	180° rotation	All

14 — **Trickle-Feed_Enable**

Project: DevSNB

Default Value: 0b

Value	Name	Description	Project
0b	Enable	Trickle Feed Enabled - Data requests are sent whenever there is space in the Display Data Buffer	All
1b	Disable	Trickle Feed Disabled - Data requests are sent in bursts.	All

13:11 — **Reserved** Project: All Format:

DVSACNTR—Video Sprite A Control Register

10 — **Tiled_Surface**

Project: All

Default Value: 0b

This bit indicates that the surface data is in tiled memory. The tile pitch is specified in bytes in the DVSASTRIDE register. Only X tiling is supported for display surfaces.

When this bit is set, it affects the hardware interpretation of the DVSASTART and DVSASURFADDR registers.

Value	Name	Description	Project
0b	Linear	Linear memory	All
1b	Tiled	Tiled memory	All

9:3 — **Reserved** Project: All Format: MBZ

2 — **Sprite_Destination_Key**

Project: All

Default Value: 0b

This bit enables the destination key function. If the pixel for the primary plane on this pipe matches the key value in DVSAKEYVAL the sprite pixel is used, otherwise the primary plane pixel is passed through the blender unmodified. Destination Key can not be enabled if source key is enabled.

Value	Name	Description	Project
0b	Disable	Destination Key is disabled	All
1b	Enable	Destination Key is enabled	All

1:0 — **Reserved** Project: All Format: MBZ

Sprite Source Pixel Format Mapping of Bits to Colors:

Note: For RGB formats, see the primary source pixel format mapping

SPRITE YUV	Y1	U	Y2	V
YUV 4:2:2 YUYV	7:0	15:8	23:16	31:24
YUV 4:2:2 UYVY	15:8	7:0	31:24	23:16
YUV 4:2:2 YVYU	7:0	31:24	23:16	15:8
YUV 4:2:2 VYUY	15:8	23:16	31:24	7:0

4.10.2 DVSALINOFF—Video Sprite A Linear Offset Register

DVSALINOFF—Video Sprite A Linear Offset Register	
Register Type:	MMIO
Address Offset:	**72184h-72187h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

Bit	Description
31:0	**Sprite_Linear_Offset** Project: All Format: This register provides the linear panning byte offset into the sprite plane. This value is added to the surface address to get the address of the first pixel to be displayed. This offset must be at least pixel aligned for unrotated RGB formats and even pixel aligned for unrotated YUV formats. When performing 180° rotation, the unpanned offset must be the difference between the last pixel of the last line of the display data in its unrotated orientation and the display surface address. When the surface is tiled, the contents of this register are ignored.

4.10.3 DVSASTRIDE—Video Sprite A Stride Register

DVSASTRIDE—Video Sprite A Stride Register	
Register Type:	MMIO
Address Offset:	**72188h-7218Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled or sprite disabled, after armed
Double Buffer Armed By:	Write to DVSASURF

Bit	Description
31:15	**Reserved** Project: All Format:
14:6	**Sprite_Stride** Project: All Format: This is the stride for the sprite in bytes. When using linear memory, this must be 64 byte aligned. When using tiled memory, this must be 512 byte aligned. This register is updated through either a command packet passed through the command stream or writes to this register. The stride is limited to a maximum of 16K bytes when sprite scaling is not enabled, 4K bytes when sprite scaling is enabled, for both linear and tiled memory.
5:0	**Reserved** Project: All Format:

4.10.4 DVSAPOS—Video Sprite A Position Register

DVSAPOS—Video Sprite A Position Register	
Register Type:	MMIO
Address Offset:	**7218Ch-7218Fh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled or sprite disabled, after armed
Double Buffer Armed By:	Write to DVSASURF, DVSAPOS write does not disarm

This register specifies the position of the sprite. Software must take care that the sprite does not extend out of the display active area. ie. Xposition + Xsize =< Xsrcsize

Bit	Description
31:28	**Reserved** Project: All Format: MBZ
27:16	**Sprite_Y-Position** Project: All Format: These 12 bits specify the vertical position in lines of the sprite (upper left corner) relative to the beginning of the active video area. When performing 180° rotation, this field specifies the vertical position of the lower right corner relative to the end of the active video area in the unrotated orientation. The defined sprite rectangle must always be completely contained within the displayable area of the screen image.
15:12	**Reserved** Project: All Format: MBZ
11:0	**Sprite_X-Position** Project: All Format: These 12 bits specify the horizontal position in pixels of the sprite (upper left corner) relative the beginning of the active video area. When performing 180° rotation, this field specifies the horizontal position of the original lower right corner relative to the original end of the active video area in the unrotated orientation. The defined sprite rectangle must always be completely contained within the displayable area of the screen image.

4.10.5 DVSASIZE—Video Sprite A Size Register

DVSASIZE—Video Sprite A Size Register	
Register Type:	MMIO
Address Offset:	**72190h-72193h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled or sprite disabled, after armed
Double Buffer Armed By:	Write to DVSASURF

This register specifies the size of the sprite. Software must take care that the sprite does not extend out of the display active area. ie. Xposition + Xsize =< Xsrcsize

Bit	Description
31:28	**Reserved** Project: All Format: MBZ
27:16	**Sprite_Height** Project: All Format: This register field is used to specify the height of the sprite in lines. The value in the register is the height minus one. The defined sprite rectangle must always be completely contained within the displayable area of the screen image.
15:12	**Reserved** Project: All Format: MBZ
11:0	**Sprite_Width** Project: All Format: This register field is used to specify the width of the sprite in pixels. This does not have to be the same as the stride but should be less than or equal to the stride in pixels. The value in the register is the width minus one. The defined sprite rectangle must always be completely contained within the displayable area of the screen image. The sprite width (actual width, not the width minus one value) is limited to even values when YUV source pixel format is used, or Pixel Multiply is set to Line/Pixel doubling or Pixel doubling.

4.10.6 DVSAKEYVAL—Video Sprite A Color Key Value Register

DVSAKEYVAL—Video Sprite A Color Key Value Register	
Register Type:	MMIO
Address Offset:	**72194h-72197h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

This register specifies the key color to be used with the mask bits to determine if the sprite source data matches the key. This register will only have an effect when the sprite color key is enabled. In source key mode this value is the minimum value for the range compare. In destination key mode this value is the compare value.

Bit	Description
31:24	**Reserved** Project: All Format: MBZ
23:16	**V_Source_Key_Min_Value/R_Source/Dest_Key_Value** Project: All Format: Specifies the color key (minimum) value for the sprite V channel source key or the Red channel source or destination key compare value.
15:8	**Y_Source_Key_Min_Value/G_Source/Dest_Key_Value** Project: All Format: Specifies the color key (minimum) value for the sprite Y channel source key or the Green channel source or destination key compare value.
7:0	**U_Source_Key_Min_Value/B_Source/Dest_Key_Value** Project: All Format: Specifies the color key (minimum) value for the sprite U channel source key or the Blue channel source or destination key compare value.

4.10.7 DVSAKEYMSK—Video Sprite A Color Key Mask Register

DVSAKEYMSK—Video Sprite A Color Key Mask Register	
Register Type:	MMIO
Address Offset:	**72198h-7219Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

For source key this register specifies which channels to perform range checking on.

For destination key this register specifies the key mask to be used with the color value bits to determine if the display source data matches the key when enabled. A zero bit in the mask indicates that the corresponding bit match failure should be ignored when determining if the pixel matches.

Note that source key and destination key are mutually exclusive modes of operation, they can not be used simultaneously. For the function that is not enabled, the associated bits in this register should be programmed to zeroes.

Bit	Description
31:27	**Reserved** Project: All Format: MBZ
26	**V/R_Channel_Source_Key_Enable** Project: All Format: Specifies the source color key enable for the V/Red channel
25	**Y/G_Channel_Source_Key_Enable** Project: All Format: Specifies the source color key enable for the Y/Green channel
24	**U/B_Channel_Source_Key_Enable** Project: All Format: Specifies the source color key enable for the U/Blue channel
23:16	**R_mask_Dest_Key_Value** Project: All Format: Specifies the destination color key mask for the sprite R channel
15:8	**G_mask_Dest_Key_Value** Project: All Format: Specifies the destination color key mask for the sprite G channel
7:0	**B_mask_Dest_Key_Value** Project: All Format: Specifies the destination color key mask for the sprite B channel

4.10.8 DVSASURF—Video Sprite A Surface Address Register

DVSASURF—Video Sprite A Surface Address Register	
Register Type:	MMIO
Address Offset:	**7219Ch-7219Fh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled
Writes to this register arm DVSA registers	

Bit	Description
31:12	**Sprite_Surface_Base_Address** Project: All Address: Graphicsdress[31:12] This address specifies the surface base address. When the surface is tiled, panning is specified using (x, y) offsets in the DVSATILEOFF register. When the surface is in linear memory, panning is specified using a linear offset in the DVSALINOFF register. This address must be 4K aligned. It represents an offset from the graphics memory aperture base and is mapped to physical pages through the global GTT. The value in this register is updated through the command streamer with synchronous flips.
11:0	**Reserved** Project: All Format:

4.10.9 DVSAKEYMAXVAL—Video Sprite A Color Key Max Value Register

DVSAKEYMAXVAL—Video Sprite A Color Key Max Value Register	
Register Type:	MMIO
Address Offset:	**721A0h-721A3h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

This register specifies the key color to be used with the mask bits to determine if the sprite source data matches the key. This register will only have an effect when the sprite source color key is enabled

Bit	Description
31:24	**Reserved** Project: All Format: MBZ
23:16	**V_Key_Max_Value** Project: All Format: Specifies the color key value for the sprite V channel
15:8	**Y_Key_Max_Value** Project: All Format: Specifies the color key value for the sprite Y channel
7:0	**U_Key_Max_Value** Project: All Format: Specifies the color key value for the sprite U channel

4.10.10 DVSATILEOFF—Video Sprite A Tiled Offset Register

DVSATILEOFF—Video Sprite A Tiled Offset Register	
Register Type:	MMIO
Address Offset:	**721A4h-721A7h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

This register specifies the panning for the sprite surface in tiled memory. The surface base address is specified in the DVSASURFADDR register, and this register is used to describe an offset from that base address. When the surface is in linear memory, the offset is specified in the DVSALINOFF register and the contents of this register are ignored. When the surface is tiled, the start position is specified in this register as an (x, y) offset from the beginning of the surface.

Bit	Description
31:28	**Reserved** Project: All Format: MBZ
27:16	**Sprite_Start_Y-Position** Project: All Format: These 12 bits specify the vertical position in lines of the beginning of the active display plane relative to the display surface. When performing 180° rotation, this field specifies the vertical position of the lower right corner relative to the start of the active display plane in the unrotated orientation.
15:12	**Reserved** Project: All Format: MBZ
11:0	**Sprite_Start_X-Position** Project: All Format: These 12 bits specify the horizontal offset in pixels of the beginning of the active display plane relative to the display surface. The offset must be even pixel aligned for unrotated YUV formats. When performing 180° rotation, this field specifies the horizontal position of the lower right corner relative to the start of the active display plane in the unrotated orientation.

4.10.11 DVSASURFLIVE—Video Sprite A Live Surface Base Address Register

DVSASURFLIVE—Video Sprite A Live Surface Base Address Register	
Register Type:	MMIO
Address Offset:	**721ACh-721AFh**
Project:	All
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32

Bit	Description
31:0	**Sprite_Surface_Base_Address** Project: All Address: GraphicsAddress[31:0] This gives the live value of the surface base address as being currently used for the plane.

4.10.12DVSASCALE—Video Sprite A Scaler Control

DVSASCALE—Video Sprite A Scaler Control

Register Type:	MMIO
Address Offset:	**72204h-72207h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled or sprite disabled, after armed
Double Buffer Armed By:	Write to DVSASURF

This register controls the sprite scaling. The DVSASIZE register gives the destination (output to pipe) size of the sprite. This register gives the source (input to sprite) size of the sprite. When scaling is enabled, the source size will be scaled up or down to the destination size.

Upscaling of any amount is allowed. Downscaling up to 16X (source/destination) is allowed. Downscaling greater than 2X will involve decimation. Downscaling increases memory bandwidth requirements. Scaling can not be used with the sprite 64bpp source pixel format. Source and destination sizes must be 3x3 (3x6 when interlacing) or greater when scaling is enabled.

Horizontal downscaling limits the maximum dot clock allowed as percent of cdclk. Rules to calculate the allowed dot clock:

Start with maximum dot clock 90% of cdclk. (There is a separate requirement that planes using 64bpp formats can not be enabled with dot clock >80% of cdclk. Subtract 10% more per horizontal decimation step (decimation steps at 2x, 4x, 8x, and 16x downscale).

Subtract 10% more if sprite is using the RGB data format.

Subtract 10% more if sprite scaling is enabled on the other pipe.

Then divide that by horizontal downscale amount within each decimation step.

The result is the maximum allowed dot clock as percent of cdclk frequency.

Example:

Scale factor	Decimation amount	YUV single pipe dot clock %	YUV dual pipe dot clock %	RGB single pipe dot clock %	RGB dual pipe dot clock %	Comment
1	1	90	80	80	70	No scaling
1.5	1	60	53	53	46	
1.99	1	45	40	40	35	Max downscale before decimation starts
2	2	80	70	70	60	
3	2	53	46	46	40	
3.99	2	40	35	35	30	
4	4	70	60	60	50	

DVSASCALE—Video Sprite A Scaler Control

6	4	46	40	40	33	
7.99	4	35	30	30	25	
8	8	60	50	50	40	
12	8	40	33	33	26	
15.99	8	30	25	25	20	Worst case dot clock
16	16	50	40	40	30	Max downscaling allowed

Bit	Description
31	**Scaling_Enable** Project: All Format: Enable Enables the scaling function. Source width can be no more than 4k bytes. For best picture quality, disable when scaling is not required. When in Self Refresh Big FIFO mode, scaling enable will be masked off while Big FIFO mode is exiting.
30:29	**Filter_Control** Project: All Default Value: 0b Filter selection

Value	Name	Description	Project
00b	Medium	Medium Filtering	All
01b	Enhancing	Edge Enhancing Filtering	All
10b	Softening	Edge Softening Filtering	All
11b	Reserved	Reserved	All

28 **Even/Odd_Field_Offset**

Project: All

Default Value: 0b

Select the vertical offset of the filtered data. Software is responsible for updating this to match the surface data.

Value	Name	Description	Project
0b	0	Vertical initial phase of 0	All
1b	0.5	Vertical initial phase of 0.5	All

DVSASCALE—Video Sprite A Scaler Control	
27	**Even/Odd_Field_Enable** Project: All Default Value: 0b Enable adjustment of the vertical offset of the filtered data. <table><tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr><tr><td>0b</td><td>Disable</td><td>Off (Vertical initial phase is 1/2 the scale factor)</td><td>All</td></tr><tr><td>1b</td><td>Enable</td><td>On (Vertical initial phase is selected by the Even/Off Field Offset bit)</td><td>All</td></tr></table>
26:16	**Source_Width** Project: All Format: The horizontal size of the source image to be scaled in pixels. Max number of pixels is 2048; minimum is 3. The value programmed is one less than the number of pixels. Source width can be no more than 4k bytes, counting from a 64 byte alignment. The sprite width (actual width, not the width minus one value) is limited to even values when YUV source pixel format is used.
15:11	**Reserved** Project: All Format: MBZ
10:0	**Source_Height** Project: All Format: The vertical size of the source image to be scaled in lines. If the source is a field, this is the number of lines in the field. Max number of lines is 2048; minimum is 3 (6 when interlacing). The value programmed is one less than the number of lines. The height must be even when sprite scaling is enabled and the pipe has set planes to interlaced fetch. That means the programmed value must be odd.

4.10.13 DVSAGAMC—Video Sprite A Gamma Correction Registers

DVSGAMC Reference Point	
Project: All	
Bit	**Description**
31:30	**Reserved** Project: All Format:
29:20	**Red Gamma Reference Point** Project: All Format:
19:10	**Green Gamma Reference Point** Project: All Format:
9:0	**Blue Gamma Reference Point** Project: All Format:

DVSGAMC Max Reference Point	
Project: All	
Bit	**Description**
31:11	**Reserved** Project: All Format:
10:0	**Final Gamma Max Reference Point** Project: All Format:

DVSAGAMC—Video Sprite A Gamma Correction Registers

Register Type:	MMIO
Address Offset:	**72300h-7234Bh**
Project:	All
Default Value:	00000000h; 04010040h; 08020080h; 0C0300C0h; 10040100h; 14050140h; 18060180h; 1C0701C0h; 20080200h; 24090240h; 280A0280h; 2C0B02C0h; 300C0300h; 340D0340h; 380E0380h; 3C0F03C0h; 00000400h; 00000400h; 00000400h;
Access:	R/W
Size (in bits):	19x32

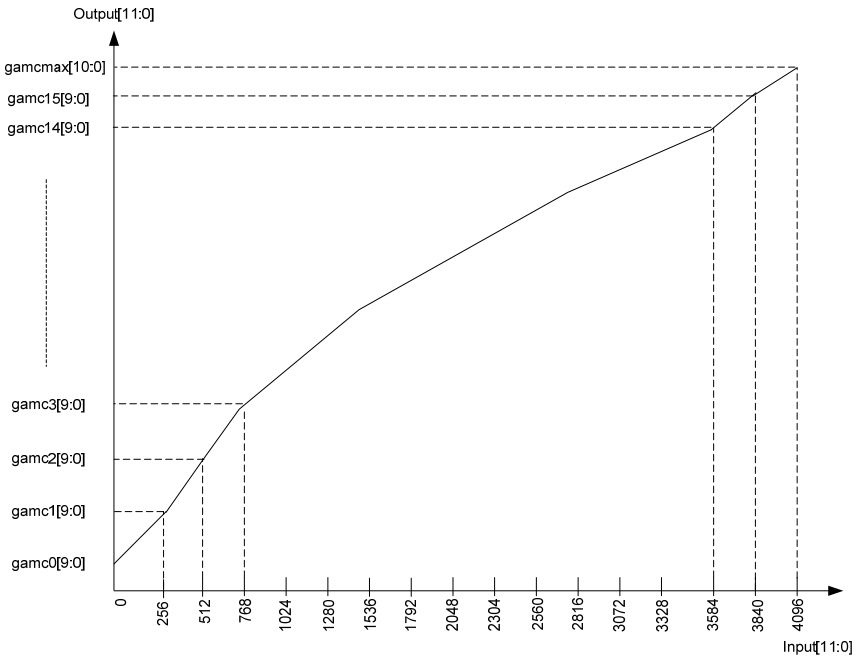
These registers are used to determine the characteristics of the gamma correction for the sprite pixel data *pre-blending*. Additional gamma correction can be done in the display pipe gamma if desired.

The gamma correction curve is represented by specifying a set of reference points spaced equally along the desired curve. Red, Green, and Blue each have 17 reference points. The first 16 reference points are 10 bit values with Red, Green, and Blue sharing a single register for each point. The final max reference point is an 11 bit value with separate registers for Red, Green, and Blue. The curve must be flat or increasing, never decreasing.

During operation the appropriate reference point pairs (adjacent) are selected for each color, and the output is interpolated between these two reference point values.

The gamma correction registers are not double-buffered. They should only be updated when the sprite is off, otherwise, screen artifacts may show.

To pass sprite pixel data through gamma correction unchanged, program the gamma reference points to the default linear ramp values. When the output from sprite is set in YUV format by programming CSC bypass, the sprite gamma correction will be bypassed.



Programming of the Piecewise-linear Estimation of Gamma Correction Curve

DVSAGAMC—Video Sprite A Gamma Correction Registers						
DWord	**Bit**	**Description**				
0	31:0	**GAMC0**	Project:	All	Format:	DVSGAMC Reference Point
1	31:0	**GAMC1**	Project:	All	Format:	DVSGAMC Reference Point
2	31:0	**GAMC2**	Project:	All	Format:	DVSGAMC Reference Point
3	31:0	**GAMC3**	Project:	All	Format:	DVSGAMC Reference Point
4	31:0	**GAMC4**	Project:	All	Format:	DVSGAMC Reference Point
5	31:0	**GAMC5**	Project:	All	Format:	DVSGAMC Reference Point
6	31:0	**GAMC6**	Project:	All	Format:	DVSGAMC Reference Point
7	31:0	**GAMC7**	Project:	All	Format:	DVSGAMC Reference Point
8	31:0	**GAMC8**	Project:	All	Format:	DVSGAMC Reference Point
9	31:0	**GAMC9**	Project:	All	Format:	DVSGAMC Reference Point
10	31:0	**GAMC10**	Project:	All	Format:	DVSGAMC Reference Point
11	31:0	**GAMC11**	Project:	All	Format:	DVSGAMC Reference Point
12	31:0	**GAMC12**	Project:	All	Format:	DVSGAMC Reference Point
13	31:0	**GAMC13**	Project:	All	Format:	DVSGAMC Reference Point
14	31:0	**GAMC14**	Project:	All	Format:	DVSGAMC Reference Point
15	31:0	**GAMC15**	Project:	All	Format:	DVSGAMC Reference Point
16	31:0	**GAMCmaxR**	Project:	All	Format:	DVSGAMC Max Reference Point
17	31:0	**GAMCmaxG**	Project:	All	Format:	DVSGAMC Max Reference Point
18	31:0	**GAMCmaxB**	Project:	All	Format:	DVSGAMC Max Reference Point

4.11 Video Sprite B Control

The DVSBCNTR, DVSBSTRIDE, DVSBPOS, DVSBSIZE, and DVSBSCALE active registers will be updated on the vertical blank or when pipe is disabled, after the DVSBSURF trigger register is written, or when the sprite B is not yet enabled – thus providing an atomic update of the video sprite B control, stride, position, size, scale, and base address registers.

4.11.1 DVSBCNTR—Video Sprite B Control Register

DVSBCNTR—Video Sprite B Control Register	
Register Type:	MMIO
Address Offset:	**73180h-73183h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled or sprite disabled, after armed
Double Buffer Armed By:	Write to DVSBSURF

Video Sprite B Plane is connected to pipe B only.

Bit	Description
31	**Sprite_Enable** Project: All Format: Enable When this bit is set, the sprite plane will generate pixels for display. When set to zero, sprite plane memory fetches cease and plane output is transparent. The display pipe must be enabled to enable the plane. When in Self Refresh Big FIFO mode, write to this register to enable the plane will be internally buffered and delayed while Big FIFO mode is exiting.
30	**Gamma_Enable** Project: All Default Value: 0b There are two gamma adjustments possible in the video sprite data path. This bit controls the gamma correction in the display pipe not the gamma control in this plane. It affects only the pixel data from this display plane. Gamma correction logic that is contained in the video sprite is disabled by loading the default values into those registers.

Value	Name	Description	Project
0b	Disable	Plane pixel data bypasses the display pipe gamma correction logic	All
1b	Enable	Plane pixel data is gamma corrected in the pipe gamma correction logic	All

Bit	Description
29	**Reserved** Project: All Format: MBZ
28	**YUV_Bypass_Excess-512_Format_Conversion** Project: All Default Value: 0b

Value	Name	Description	Project
0b	Disable	Disable excess-512 conversion	All
1b	Enable	Enable excess-512 conversion	All

DVSBCNTR—Video Sprite B Control Register

27 — **Range_Correction_Disable**

Project: All

Default Value: 0b

Setting this bit disables the YUV range correction logic. Normally the range compressed YUV is expanded to full range RGB, setting this bit will generate range compressed RGB. This bit should also be used if full range YUV source material is used. This bit has no effect on RGB source formats.

Value	Name	Description	Project
0b	Enable	Range correction enabled	All
1b	Disable	No range correction	All

26:25 — **Source_Pixel_Format**

Project: DevSNB

Default Value: 0b

This field selects the pixel format for the sprite. Before entering the blender, each source format is converted to 12 bits per pixel.

Value	Name	Description	Project
00b	YUV 4:2:2	YUV 4:2:2 packed pixel format (byte order programmed separately)	All
01b	RGB 32-bit 2:10:10:10	RGB 32-bit 2:10:10:10 pixel format (color order programmed separately). Ignore alpha.	All
10b	RGB 32-bit 8:8:8:8	RGB 32-bit 8:8:8:8 pixel format (color order programmed separately). Ignore alpha.	All
11b	RGB 64-bit 16:16:16:16 Use of 64bpp format will limit the maximum dot clock to 80% of cdclk.	RGB 64-bit 16:16:16:16 floating point pixel format (color order programmed separately). Ignore alpha.	All

24 — **Pipe_Color_Space_Conversion_Enable**

Project: All

Default Value: 0b

This bit enables pipe color space conversion for the plane pixel data. This is separate from the color conversion logic within the sprite plane. CSC mode in the pipe CSC registers must be set to match the format of the plane pixel data after the color conversion logic within the sprite plane.

Value	Name	Description	Project
0b	Disable	Plane pixel data bypasses the pipe color space conversion logic	All
1b	Enable	Plane pixel data passes through the pipe color space conversion logic	All

23 — **Reserved** Project: All Format: MBZ

DVSBCNTR—Video Sprite B Control Register

22	**Sprite_Source_Key_Enable** Project: All Default Value: 0b This bit enables source color keying. Sprite pixel values that match (within range) the key will become transparent. Source key can not be enabled if destination key is enabled.

Value	Name	Description	Project
0b	Disable	Sprite source key is disabled	All
1b	Enable	Sprite source key is enabled	All

22	**Sprite_Source_Key_Enable** Project: All Default Value: 0b This bit enables source color keying. Sprite pixel values that match (within range) the key will become transparent. Source key can not be enabled if destination key is enabled.

Value	Name	Description	Project
0b		Sprite source key is disabled	All
1b		Sprite source key is enabled	All

21	**Reserved** Project: DevSNB
20	**RGB_Color_Order** Project: DevSNB Default Value: 0b This field is used to select the color order when using RGB data formats. For other formats, this field is ignored.

Value	Name	Description	Project
0b	BGRX	BGRX (MSB-X:R:G:B)	All
1b	RGBX	RGBX (MSB-X:B:G:R)	All

20	**Reserved** Project: DevILK Format:

DVSBCNTR—Video Sprite B Control Register

19 — **Color_Conversion_Disabled**

Project: All

Default Value: 0b

This bit enables or disables the color conversion logic internal to the sprite. Color conversion is intended to be used with the formats that support YUV format. This bit is ignored when using RGB source formats.

Value	Name	Description	Project
0b	Enable	Pixel data is sent through the sprite color conversion logic (only applies to YUV formats)	All
1b	Disable	Pixel data is not sent through the sprite YUV->RGB color conversion logic.	All

18 — **YUV_Format**

Project: All

Default Value: 0b

This bit specifies the source YUV format for the YUV to RGB color conversion operation. This field is ignored when source data is RGB.

Value	Name	Description	Project
0b	BT.601	ITU-R Recommendation BT.601	All
1b	BT.709	ITU-R Recommendation BT.709	All

17:16 — **YUV_Byte_Order**

Project: All

Default Value: 0b

This field is used to select the byte order when using YUV 4:2:2 data formats. For other formats, this field is ignored.

Value	Name	Description	Project
00b	YUYV	YUYV (8:8:8:8 MSB-V:Y_2:U:Y_1)	All
01b	UYVY	UYVY (8:8:8:8 MSB-Y_2:V:Y_1:U)	All
10b	YVYU	YVYU (8:8:8:8 MSB-U:Y_2:V:Y_1)	All
11b	VYUY	VYUY (8:8:8:8 MSB-Y_2:U:Y_1:V)	All

DVSBCNTR—Video Sprite B Control Register

15 — **180°_Display_Rotation**

Project: All

Default Value: 0b

This mode causes the plane to be rotated 180°. In addition to setting this bit, software must also set the surface address offset to the lower right corner of the unrotated and unscaled image and calculate the x, y offset as relative to the lower right corner.

Value	Name	Description	Project
0b	None	No rotation	All
1b	180	180° rotation	All

14 — **Trickle-Feed_Enable**

Project: DevSNB

Default Value: 0b

Value	Name	Description	Project
0b	Enable	Trickle Feed Enabled - Data requests are sent whenever there is space in the Display Data Buffer	All
1b	Disable	Trickle Feed Disabled - Data requests are sent in bursts.	All

13:11 — **Reserved** Project: All Format:

10 — **Tiled_Surface**

Project: All

Default Value: 0b

This bit indicates that the surface data is in tiled memory. The tile pitch is specified in bytes in the DVSBSTRIDE register. Only X tiling is supported for display surfaces.

When this bit is set, it affects the hardware interpretation of the DVSBSTART and DVSBSURFADDR registers.

Value	Name	Description	Project
0b	Linear	Linear memory	All
1b	Tiled	Tiled memory	All

9:3 — **Reserved** Project: All Format: MBZ

DVSBCNTR—Video Sprite B Control Register	
2	**Sprite_Destination_Key** Project: All Default Value: 0b This bit enables the destination key function. If the pixel for the primary plane on this pipe matches the key value in DVSBKEYVAL the sprite pixel is used, otherwise the primary plane pixel is passed through the blender unmodified. Destination Key can not be enabled if source key is enabled.

Value	Name	Description	Project
0b	Disable	Destination Key is disabled	All
1b	Enable	Destination Key is enabled	All

1:0	**Reserved** Project: All Format: MBZ

See DVSACNTR - Sprite Source Pixel Format Mapping of Bits to Colors

4.11.2 DVSBLINOFF—Video Sprite B Linear Offset Register

DVSBLINOFF—Video Sprite B Linear Offset Register	
Register Type:	MMIO
Address Offset:	**73184h-73187h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

Bit	Description
31:0	**Sprite_Linear_Offset** Project: All Format: This register provides the linear panning byte offset into the sprite plane. This value is added to the surface address to get the address of the first pixel to be displayed. This offset must be at least pixel aligned for unrotated RGB formats and even pixel aligned for unrotated YUV formats. When performing 180° rotation, the unpanned offset must be the difference between the last pixel of the last line of the display data in its unrotated orientation and the display surface address. When the surface is tiled, the contents of this register are ignored.

4.11.3 DVSBSTRIDE—Video Sprite B Stride Register

DVSBSTRIDE—Video Sprite B Stride Register	
Register Type:	MMIO
Address Offset:	**73188h-7318Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled or sprite disabled, after armed
Double Buffer Armed By:	Write to DVSBSURF

Bit	Description
31:15	**Reserved** Project: All Format:
14:6	**Sprite_Stride** Project: All Format: This is the stride for the sprite in bytes. When using linear memory, this must be 64 byte aligned. When using tiled memory, this must be 512 byte aligned. This register is updated through either a command packet passed through the command stream or writes to this register. The stride is limited to a maximum of 16K bytes when sprite scaling is not enabled, 4K bytes when sprite scaling is enabled, for both linear and tiled memory.
5:0	**Reserved** Project: All Format:

4.11.4 DVSBPOS—Video Sprite B Position Register

DVSBPOS—Video Sprite B Position Register	
Register Type:	MMIO
Address Offset:	**7318Ch-7318Fh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled or sprite disabled, after armed
Double Buffer Armed By:	Write to DVSBSURF, DVSAPOS write does not disarm

This register specifies the position of the sprite. Software must take care that the sprite does not extend out of the display active area. ie. Xposition + Xsize =< Xsrcsize

Bit	Description
31:28	**Reserved** Project: All Format: MBZ
27:16	**Sprite_Y-Position** Project: All Format: These 12 bits specify the vertical position in lines of the sprite (upper left corner) relative to the beginning of the active video area. When performing 180° rotation, this field specifies the vertical position of the lower right corner relative to the end of the active video area in the unrotated orientation. The defined sprite rectangle must always be completely contained within the displayable area of the screen image.

DVSBPOS—Video Sprite B Position Register						
15:12	**Reserved**	Project:	All		Format:	MBZ
11:0	**Sprite_X-Position**	Project:	All	Format:		
	These 12 bits specify the horizontal position in pixels of the sprite (upper left corner) relative the beginning of the active video area. When performing 180° rotation, this field specifies the horizontal position of the original lower right corner relative to the original end of the active video area in the unrotated orientation. The defined sprite rectangle must always be completely contained within the displayable area of the screen image.					

4.11.5 DVSBSIZE—Video Sprite B Size Register

DVSBSIZE—Video Sprite B Size Register	
Register Type:	MMIO
Address Offset:	**73190h-73193h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled or sprite disabled, after armed
Double Buffer Armed By:	Write to DVSBSURF

This register specifies the size of the sprite. Software must take care that the sprite does not extend out of the display active area. ie. Xposition + Xsize =< Xsrcsize

Bit	Description					
31:28	**Reserved**	Project:	All		Format:	MBZ
27:16	**Sprite_Height**	Project:	All	Format:		
	This register field is used to specify the height of the sprite in lines. The value in the register is the height minus one. The defined sprite rectangle must always be completely contained within the displayable area of the screen image.					
15:12	**Reserved**	Project:	All		Format:	MBZ
11:0	**Sprite_Width**	Project:	All	Format:		
	This register field is used to specify the width of the sprite in pixels. This does not have to be the same as the stride but should be less than or equal to the stride in pixels. The value in the register is the width minus one. The defined sprite rectangle must always be completely contained within the displayable area of the screen image. The sprite width (actual width, not the width minus one value) is limited to even values when YUV source pixel format is used, or Pixel Multiply is set to Line/Pixel doubling or Pixel doubling.					

4.11.6 DVSBKEYVAL—Video Sprite B Color Key Value Register

DVSBKEYVAL—Video Sprite B Color Key Value Register	
Register Type:	MMIO
Address Offset:	**73194h-73197h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

This register specifies the key color to be used with the mask bits to determine if the sprite source data matches the key. This register will only have an effect when the sprite color key is enabled. In source key mode this value is the minimum value for the range compare. In destination key mode this value is the compare value.

Bit	Description
31:24	**Reserved** Project: All Format: MBZ
23:16	**V_Source_Key_Min_Value/R_Source/Dest_Key_Value** Project: All Format: Specifies the color key (minimum) value for the sprite V channel source key or the Red channel source or destination key compare value.
15:8	**Y_Source_Key_Min_Value/G_Source/Dest_Key_Value** Project: All Format: Specifies the color key (minimum) value for the sprite Y channel source key or the Green channel source or destination key compare value.
7:0	**U_Source_Key_Min_Value/B_Source/Dest_Key_Value** Project: All Format: Specifies the color key (minimum) value for the sprite U channel source key or the Blue channel source or destination key compare value.

4.11.7 DVSBKEYMSK—Video Sprite B Color Key Mask Register

DVSBKEYMSK—Video Sprite B Color Key Mask Register	
Register Type:	MMIO
Address Offset:	**73198h-7319Bh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

For source key this register specifies which channels to perform range checking on.

For destination key this register specifies the key mask to be used with the color value bits to determine if the display source data matches the key when enabled. A zero bit in the mask indicates that the corresponding bit match failure should be ignored when determining if the pixel matches.

Note that source key and destination key are mutually exclusive modes of operation, they can not be used simultaneously. For the function that is not enabled, the associated bits in this register should be programmed to zeroes.

Bit	Description
31:27	**Reserved** Project: All Format: MBZ
26	**V/R_Channel_Source_Key_Enable** Project: All Format: Specifies the source color key enable for the V/Red channel
25	**Y/G_Channel_Source_Key_Enable** Project: All Format: Specifies the source color key enable for the Y/Green channel
24	**U/B_Channel_Source_Key_Enable** Project: All Format: Specifies the source color key enable for the U/Blue channel
23:16	**R_mask_Dest_Key_Value** Project: All Format: Specifies the destination color key mask for the sprite R channel
15:8	**G_mask_Dest_Key_Value** Project: All Format: Specifies the destination color key mask for the sprite G channel
7:0	**B_mask_Dest_Key_Value** Project: All Format: Specifies the destination color key mask for the sprite B channel

4.11.8 DVSBSURF—Video Sprite B Surface Address Register

DVSBSURF—Video Sprite B Surface Address Register	
Register Type:	MMIO
Address Offset:	**7319Ch-7319Fh**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

Writes to this register arm DVSB registers

Bit	Description
31:12	**Sprite_Surface_Base_Address** Project: All Address: GraphicsAddress[31:12] This address specifies the surface base address. When the surface is tiled, panning is specified using (x, y) offsets in the DVSBTILEOFF register. When the surface is in linear memory, panning is specified using a linear offset in the DVSBLINOFF register. This address must be 4K aligned. It represents an offset from the graphics memory aperture base and is mapped to physical pages through the global GTT. The value in this register is updated through the command streamer with synchronous flips.
11:0	**Reserved** Project: All Format:

4.11.9 DVSBKEYMAXVAL—Video Sprite B Color Key Max Value Register

DVSBKEYMAXVAL—Video Sprite B Color Key Max Value Register	
Register Type:	MMIO
Address Offset:	**731A0h-731A3h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled

This register specifies the key color to be used with the mask bits to determine if the sprite source data matches the key. This register will only have an effect when the sprite source color key is enabled.

Bit	Description
31:24	**Reserved** Project: All Format: MBZ
23:16	**V_Key_Max_Value** Project: All Format: Specifies the color key value for the sprite V channel

DVSBKEYMAXVAL—Video Sprite B Color Key Max Value Register				
15:8	**Y_Key_Max_Value** Specifies the color key value for the sprite Y channel	Project:	All	Format:
7:0	**U_Key_Max_Value** Specifies the color key value for the sprite U channel	Project:	All	Format:

4.11.10DVSBTILEOFF—Video Sprite B Tiled Offset Register

DVSBTILEOFF—Video Sprite B Tiled Offset Register	
Register Type:	MMIO
Address Offset:	**731A4h-731A7h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Double Buffer Update Point:	Start of vertical blank or pipe disabled

This register specifies the panning for the sprite surface in tiled memory. The surface base address is specified in the DVSBSURFADDR register, and this register is used to describe an offset from that base address. When the surface is in linear memory, the offset is specified in the DVSBLINOFF register and the contents of this register are ignored. When the surface is tiled, the start position is specified in this register as an (x, y) offset from the beginning of the surface.

Bit	Description			
31:28	**Reserved**	Project: All	Format: MBZ	
27:16	**Sprite_Start_Y-Position** These 12 bits specify the vertical position in lines of the beginning of the active display plane relative to the display surface. When performing 180° rotation, this field specifies the vertical position of the lower right corner relative to the start of the active display plane in the unrotated orientation.	Project:	All	Format:
15:12	**Reserved**	Project: All	Format: MBZ	
11:0	**Sprite_Start_X-Position** These 12 bits specify the horizontal offset in pixels of the beginning of the active display plane relative to the display surface. The offset must be even pixel aligned for unrotated YUV formats. When performing 180° rotation, this field specifies the horizontal position of the lower right corner relative to the start of the active display plane in the unrotated orientation.	Project:	All	Format:

4.11.11DVSBSURFLIVE—Video Sprite B Live Surface Base Address Register

DVSBSURFLIVE—Video Sprite B Live Surface Base Address Register	
Register Type:	MMIO
Address Offset:	**731ACh-731AFh**
Project:	All
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32

Bit	Description
31:0	**Sprite_Surface_Base_Address** Project: All Address: GraphicsAddress[31:0] This gives the live value of the surface base address as being currently used for the plane.

4.11.12DVSBSCALE—Video Sprite B Scaler Control

DVSBSCALE—Video Sprite B Scaler Control	
Register Type:	MMIO
Address Offset:	**73204h-73207h**
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe disabled or sprite disabled, after armed
Double Buffer Armed By:	Write to DVSBSURF

This register controls the sprite scaling. The DVSBSIZE register gives the destination (output to pipe) size of the sprite. This register gives the source (input to sprite) size of the sprite. When scaling is enabled, the source size will be scaled up or down to the destination size.

Upscaling of any amount is allowed. Downscaling up to 16X (source/destination) is allowed. Downscaling greater than 2X will involve decimation. Downscaling increases memory bandwidth requirements. Scaling can not be used with the sprite 64bpp source pixel format. Source and destination sizes must be 3x3 (3x6 when interlacing) or greater when scaling is enabled.

Horizontal downscaling limits the maximum dot clock allowed as percent of cdclk. See DVSASCALE for the rules to calculate the allowed dot clock.

Bit	Description
31	**Scaling_Enable** Project: All Format: Enable Enables the scaling function. Source width can be no more than 4k bytes. For best picture quality, disable when scaling is not required. When in Self Refresh Big FIFO mode, scaling enable will be masked off while Big FIFO mode is exiting.

DVSBSCALE—Video Sprite B Scaler Control

30:29 Filter_Control

Project: All

Default Value: 0b

Filter selection

Value	Name	Description	Project
00b	Medium	Medium Filtering	All
01b	Enhancing	Edge Enhancing Filtering	All
10b	Softening	Edge Softening Filtering	All
11b	Reserved	Reserved	All

28 Even/Odd_Field_Offset

Project: All

Default Value: 0b

Select the vertical offset of the filtered data. Software is responsible for updating this to match the surface data.

Value	Name	Description	Project
0b	0	Vertical initial phase of 0	All
1b	0.5	Vertical initial phase of 0.5	All

27 Even/Odd_Field_Enable

Project: All

Default Value: 0b

Enable adjustment of the vertical offset of the filtered data.

Value	Name	Description	Project
0b	Disable	Off (Vertical initial phase is 1/2 the scale factor)	All
1b	Enable	On (Vertical initial phase is selected by the Even/Off Field Offset bit)	All

26:16 Source_Width Project: All Format:

The horizontal size of the source image to be scaled in pixels. Max number of pixels is 2048; minimum is 3. The value programmed is one less than the number of pixels. Source width can be no more than 4k bytes, counting from a 64 byte alignment. The sprite width (actual width, not the width minus one value) is limited to even values when YUV source pixel format is used.

15:11 Reserved Project: All Format: MBZ

10:0 Source_Height Project: All Format:

The vertical size of the source image to be scaled in lines. If the source is a field, this is the number of lines in the field. Max number of lines is 2048; minimum is 3 (6 when interlacing). The value programmed is one less than the number of lines.

The height must be even when sprite scaling is enabled and the pipe has set planes to interlaced fetch. That means the programmed value must be odd.

4.11.13 DVSBGAMC—Video Sprite B Gamma Correction Registers

DVSBGAMC—Video Sprite B Gamma Correction Registers

Register Type: MMIO
Address Offset: 73300h
Project: All
Default Value: 00000000h; 04010040h; 08020080h; 0C0300C0h; 10040100h; 14050140h; 18060180h; 1C0701C0h; 20080200h; 24090240h; 280A0280h; 2C0B02C0h; 300C0300h; 340D0340h; 380E0380h; 3C0F03C0h; 00000400h; 00000400h; 00000400h;
Access: R/W
Size (in bits): 19x32

See Video Sprite A description

DWord	Bit	Description				
0	31:0	**GAMC0**	Project:	All	Format:	DVSGAMC Reference Point
1	31:0	**GAMC1**	Project:	All	Format:	DVSGAMC Reference Point
2	31:0	**GAMC2**	Project:	All	Format:	DVSGAMC Reference Point
3	31:0	**GAMC3**	Project:	All	Format:	DVSGAMC Reference Point
4	31:0	**GAMC4**	Project:	All	Format:	DVSGAMC Reference Point
5	31:0	**GAMC5**	Project:	All	Format:	DVSGAMC Reference Point
6	31:0	**GAMC6**	Project:	All	Format:	DVSGAMC Reference Point
7	31:0	**GAMC7**	Project:	All	Format:	DVSGAMC Reference Point
8	31:0	**GAMC8**	Project:	All	Format:	DVSGAMC Reference Point
9	31:0	**GAMC9**	Project:	All	Format:	DVSGAMC Reference Point
10	31:0	**GAMC10**	Project:	All	Format:	DVSGAMC Reference Point
11	31:0	**GAMC11**	Project:	All	Format:	DVSGAMC Reference Point
12	31:0	**GAMC12**	Project:	All	Format:	DVSGAMC Reference Point
13	31:0	**GAMC13**	Project:	All	Format:	DVSGAMC Reference Point
14	31:0	**GAMC14**	Project:	All	Format:	DVSGAMC Reference Point
15	31:0	**GAMC15**	Project:	All	Format:	DVSGAMC Reference Point
16	31:0	**GAMCmaxR**	Project:	All	Format:	DVSGAMC Max Reference Point
17	31:0	**GAMCmaxG**	Project:	All	Format:	DVSGAMC Max Reference Point
18	31:0	**GAMCmaxB**	Project:	All	Format:	DVSGAMC Max Reference Point

Revision History

Revision Number	Description	Revision Date
1.0	First 2011 OpenSource edition	May 2011

§§

Intel® OpenSource HD Graphics Programmer's Reference Manual (PRM)

Volume 3 Part 3: PCH Display Registers (SandyBridge)

For the 2011 Intel Core Processor Family

May 2011

Revision 1.0

NOTICE:

This document contains information on products in the design phase of development, and Intel reserves the right to add or remove product features at any time, with or without changes to this open source documentation.

Doc Ref #: IHD-OS-V3 Pt3 – 05 11

Creative Commons License

You are free to Share — to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

No Derivative Works. You may not alter, transform, or build upon this work.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The SandyBridge chipset family, Havendale/Auburndale chipset family, Intel® 965 Express Chipset Family, Intel® G35 Express Chipset, and Intel® 965GMx Chipset Mobile Family Graphics Controller may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel® sales office or your distributor to obtain the latest specifications and before placing your product order. I2C is a two-wire communications bus/protocol developed by Philips. SMBus is a subset of the I2C bus/protocol and was developed by Intel®. Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

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

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

Copyright © 2011, Intel Corporation. All rights reserved.

Contents

1. South Display Engine Registers [DevCPT] 5
1.1 Introduction 5
1.1.1 Terminology 5
1.1.2 Display Mode Set Sequence 6
1.1.3 Simultaneous Display Capabilities on a Single Display Pipe/Transcoder 6
1.1.4 Register Instances and Address Offsets 7
2. South Shared Functions (C0000h–CFFFFh) 8
2.1 South Display Engine Interrupt Control Registers 8
2.1.1 South Display Engine Interrupt Bit Definition 8
2.1.2 ISR — Interrupt Status 10
2.1.3 IMR — Interrupt Mask 11
2.1.4 IIR — Interrupt Identity 12
2.1.5 IER — Interrupt Enable 13
2.1.6 SHOTPLUG_CTL — South Hot Plug Control 13
2.2 GMBUS and I/O Control Registers (C5000h–C5FFFh) 16
2.2.1 GPIO Pin Usage (By Functions) 16
2.2.2 GPIO_CTL –GPIO Control 18
2.2.3 GMBUS Controller Programming Interface 21
2.3 Display Clock Control Registers (C6000h–C6FFFH) 32
2.3.1 DPLL_CTL—DPLL Control 34
2.3.2 DPLL_FP0—DPLL Divisor 0 37
2.3.3 DPLL_FP1—DPLL Divisor 1 39
2.3.4 DREF_CONTROL — Display Reference Clock Control Register 40
2.3.5 RAWCLK_FREQ—Rawclk Frequency 43
2.3.6 SSC4PARMS – SSC4 Parameters 43
2.3.7 DPLL_SEL— DPLL Select 45
2.4 Panel Power Sequencing Registers 47
2.4.1 PP_STATUS—Panel Power Status Register 47
2.4.2 PP_CONTROL—Panel Power Control Register 49
2.4.3 PP_ON_DELAYS—Panel Power On Sequencing Delays 51
2.4.4 PP_OFF_DELAYS—Panel Power Off Sequencing Delays 52
2.4.5 PP_DIVISOR—Panel Power Cycle Delay and Reference Divisor 53
2.5 Backlight Control Registers 54
2.5.1 Backlight PWM PCH Control Register 1 54
2.5.2 SBLC_BLM_CTL2—South BLM Control 2 55
3. South Display Engine Transcoder and Port Controls 56
3.1 Transcoder Timing 56
3.1.1 HTOTAL—Horizontal Total 56
3.1.2 HBLANK—Horizontal Blank 57
3.1.3 HSYNC—Horizontal Sync 58
3.1.4 VTOTAL—Vertical Total 59
3.1.5 VBLANK—Vertical Blank 60
3.1.6 VSYNC—Vertical Sync 61
3.1.7 VSYNCSHIFT— Vertical Sync Shift 62
3.2 Transcoder M/N Values 62
3.2.1 DATAM— Data M Value 63
3.2.2 DATAN— Data N Value 63
3.2.3 LINKM— Link M Value 64
3.2.4 LINKN— Link N Value 64

3.3 Transcoder Video DIP 65
3.3.1 VIDEO_DIP_CTL—Video DIP Control 65
3.3.2 VIDEO_DIP_DATA–Video Data Island Packet Data 69
3.3.3 VIDEO_DIP_GCP–Video Data Island Payload 70
3.4 Transcoder DisplayPort Control 72
3.4.1 TRANS_DP_CTL—Transcoder DisplayPort Control 72
3.5 Analog Port CRT DAC 74
3.5.1 DAC_CTL—Analog Port CRT DAC Control 74
3.6 HDMI Port 77
3.6.1 HDMI_CTL—HDMI Port Control 77
3.7 LVDS 81
3.7.1 LVDS—LVDS Port Control Register 81
3.8 DisplayPort 86
3.8.1 DP_CTL—DisplayPort Control 86
3.8.2 DP_AUX_CTL—DisplayPort AUX Channel Control 89
3.8.3 DP_AUX_DATA—DisplayPort AUX Channel Data 91
3.8.4 DP_BUFTRANS—DisplayPort Buffer Translation 92
4. South Display Engine Audio 95
4.1 Audio Programming Sequence 95
4.2 Audio Configuration 96
4.2.1 AUD_CONFIG—Audio Configuration 96
4.2.2 AUD_CTS_ENABLE – Audio CTS Programming Enable 98
4.2.3 AUD_MISC_CTRL—Audio MISC Control 99
4.2.4 AUD_VID_DID—Audio Vendor ID / Device ID 100
4.2.5 AUD_RID—Audio Revision ID 100
4.2.6 AUD_PWRST—Audio Power State 101
4.2.7 AUD_PINW_CONNLNG_LIST—Audio Connection List 103
4.2.8 AUD_PINW_CONNLNG_SEL—Audio Connection Select 103
4.2.9 AUD_CNTL_ST—Audio Control State 104
4.2.10 AUD_CNTRL_ST2— Audio Control State 2 107
4.2.11 AUD_HDMIW_HDMIEDID—Audio HDMI Data EDID Block 109
4.2.12 AUD_HDMIW_INFOFR—Audio Widget Data Island Packet 110
5. South Display Engine Transcoder and FDI Control 111
5.1 Transcoder Control 111
5.1.1 TRANS_CONF—Transcoder Configuration 111
5.2 FDI Receiver 113
5.2.1 FDI_RX_CTL— FDI Rx Control 113
5.2.2 FDI_RX_MISC— FDI Rx Miscellaneous 118
5.2.3 FDI_RX_IMR — FDI Rx Interrupt Mask 119
5.2.4 FDI_RX_IIR — FDI Rx Interrupt Identity 121
5.2.5 FDI_RX_TUSIZE— FDI Rx Transfer Unit Size 121

1. South Display Engine Registers [DevCPT]

1.1 Introduction

This chapter contains the register descriptions for the display portion of a family of graphics devices. These registers vary by devices within the family of devices, so special attention needs to be paid to which devices use which registers and register fields.

Different devices within the family may add, modify, or delete registers or register fields relative to another device in the same family based on the supported functions of that device.

1.1.1 Terminology

Description	Software Use	Should be implemented as
Read/Write, R/W	This bit can be read or written.	
Reserved	Don't assume a value for these bits. Writes have no effect.	Writes are ignored. Reads return zero.
Reserved: must be zero, MBZ	Software must always write a zero to these bits. This allows new features to be added using these bits that will be disabled when using old software and as the default case.	Writes are ignored. Reads return zero. Maybe be connected as Read/Write in future projects.
Reserved: PBC, software must preserve contents	Software must write the original value back to this bit. This allows new features to be added using these bits.	Read only.
Read Only	This bit is read only. The read value is determined by hardware. Writes to this bit have no effect.	According to each specific bit. The bit value is determined by hardware and not affected by register writes to the actual bit.
Read/Clear, Read/Write Clear	This bit can be read. Writes to it with a one cause the bit to clear.	Hardware events cause the bit to be set and the bit will be cleared on a write operation where the corresponding bit has a one for a value.
Double Buffered	Write when desired. Read gives the unbuffered value (written value) unless specified otherwise. Written values will update to take effect after a certain point. Some have a specific arming sequence where a write to another register is required before the update can take place. This is used to ensure atomic updates of several registers.	Two stages of registers used. First stage is written into and used for readback (unless specified otherwise). First stage value is transferred into second stage at the update point. Second stage value is used to control hardware. Arm/disarm flag for specific arming sequences.

1.1.2 Display Mode Set Sequence

See the North Display Engine Registers document.

1.1.3 Simultaneous Display Capabilities on a Single Display Pipe/Transcoder

	Embedded DP (on CPU)	Inte-grated LVDS	HDMI	DP	CRT
Embedded DP (on CPU)		No (7)	No (7)	No (7)	No (7)
Integrated LVDS			No (2, 3, 8)	No (2, 8,9)	No (3, 8)
HDMI			No(6, 8)	No(6, 8)	No (5, 8)
DP				No(3, 6, 8)	No (5, 8)
CRT					

Shading: Rose = Does not work, Yellow = Some cases work

1. No internal LVDS and HDMI, or DP on same pipe/transcoder.
2. No SSC on CRT, DVI, or HDMI. DP optionally has SSC.
3. Only works if DP/HDMI is in 24bpp mode.
4. Digital ports are multiplexed on the same pins, only works if ports are different.
5. Embedded DP is on the CPU; can not share the link.
6. Dithering, range correction, and gamma are done in the CPU; the display with lower bpp can truncate or the display with higher bpp can lose bits. One of the displays dictates range and gamma.
7. No DisplayPort allowed with other port on the same pipe/transcoder.
8. No HDMI allowed with another HDMI on the same transcoder.

1.1.4 Register Instances and Address Offsets

The main body of the register document only contains generic register format information.

The register offset spreadsheet gives the address of each register and which format it is an instance of.

2. South Shared Functions (C0000h–CFFFFh)

2.1 South Display Engine Interrupt Control Registers

2.1.1 South Display Engine Interrupt Bit Definition

South Display Engine Interrupt Bit Definition				
Project: All				
South Display Engine (SDE) interrupt bits come from events within the south display engine. The SDE_IIR bits are ORed together to generate the South/PCH Display Interrupt Event which will appear in the North Display Engine Interrupt Control Registers. Only the rising edge of the PCH Display interrupt will cause the North Display IIR (DEIIR) PCH Display Interrupt event bit to be set, so all PCH Display Interrupts, including back to back interrupts, must be cleared in the SDEIIR before a new PCH Display Interrupt can cause the DEIIR to be set. The South Display Engine Interrupt Control Registers all share the same bit definitions from this table.				
Bit	**Description**			
31	**Audio_Power_State_change_Port_D** This is an active high pulse when there is a power state change for audio for port D.	Project:	All	Format:
30	**Audio_Power_State_change_Port_C** This is an active high pulse when there is a power state change for audio for port C.	Project:	All	Format:
29	**Audio_Power_State_change_Port_B** This is an active high pulse when there is a power state change for audio for port B.	Project:	All	Format:
28	**Reserved**	Project:	All	Format:
27	**AUX_Channel_D** This is an active high pulse on the AUX D done event	Project:	All	Format:
26	**AUX_Channel_C** This is an active high pulse on the AUX C done event	Project:	All	Format:
25	**AUX_Channel_B** This is an active high pulse on the AUX B done event	Project:	All	Format:
24	**Reserved**	Project:	All	Format:

South Display Engine Interrupt Bit Definition	
23	**DisplayPort/HDMI/DVI_D_Hotplug** Project: All Format: The ISR is an active high level representing the Digital Port D hotplug line when the Digital Port D hotplug detect input is enabled. The unmasked IIR is set on either a short or long pulse detection status in the Digital Port Hot Plug Control Register.
22	**DisplayPort/HDMI/DVI_C_Hotplug** Project: All Format: The ISR is an active high level representing the Digital Port C hotplug line when the Digital Port C hotplug detect input is enabled. The unmasked IIR is set on either a short or long pulse detection status in the Digital Port Hot Plug Control Register.
21	**DisplayPort/HDMI/DVI_B_Hotplug** Project: All Format: The ISR is an active high level representing the Digital Port B hotplug line when the Digital Port B hotplug detect input is enabled. The unmasked IIR is set on either a short or long pulse detection status in the Digital Port Hot Plug Control Register.
20	**Reserved** Project: All Format:
19	**CRT_Hotplug** Project: All Format: The ISR is an active high level representing the ORed together blue and green channel detection status as of the last detection cycle. The unmasked IIR is set on the rising or falling edges of the blue or green channel detection status in the Analog Port CRT DAC Control Register.
18	**Reserved** Project: All
17	**Gmbus** Project: All Format: This is an active high pulse when any of the events unmasked events in GMBUS4 Interrupt Mask register occur.
16	**Reserved** Project: All
15:11	**Reserved** Project: All Format:
10	**Audio_CP_Request_Transcoder_C** Project: All Format: This is an active high level indicating content protection is requested by audio azalia verb programming for transcoder C. It is valid after the Audio_CP_Change_Transcoder_C event has occurred.
9	**Audio_CP_Change_Transcoder_C** Project: All Format: This is an active high pulse when there is a change in the protection request from audio azalia verb programming for transcoder C.
8	**FDI_RX_Interrupts_Combined_C** Project: All Format: This is an active high level while any of the FDI_RX_ISR bits are set for transcoder C
7	**Reserved** Project: All Format:
6	**Audio_CP_Request_Transcoder_B** Project: All Format: This is an active high level indicating content protection is requested by audio azalia verb programming for transcoder B. It is valid after the Audio_CP_Change_Transcoder_B event has occurred.
5	**Audio_CP_Change_Transcoder_B** Project: All Format: This is an active high pulse when there is a change in the protection request from audio azalia verb programming for transcoder B.

South Display Engine Interrupt Bit Definition				
4	**FDI_RX_Interrupts_Combined_B** This is an active high level while any of the FDI_RX_ISR bits are set for transcoder B	Project:	All	Format:
3	**Reserved**	Project:	All	Format:
2	**Audio_CP_Request_Transcoder_A** This is an active high level indicating content protection is requested by audio azalia verb programming for transcoder A. It is valid after the Audio_CP_Change_Transcoder_A event has occurred.	Project:	All	Format:
1	**Audio_CP_Change_Transcoder_A** This is an active high pulse when there is a change in the protection request from audio azalia verb programming for transcoder A.	Project:	All	Format:
0	**FDI_RX_Interrupts_Combined_A** This is an active high level while any of the FDI_RX_ISR bits are set for transcoder A	Project:	All	Format:

2.1.2 ISR — Interrupt Status

ISR

Register Type: MMIO
Project: All
Default Value: 00000000h
Access: Read Only
Size (in bits): 32

See the interrupt bit definition tables to find the source event for each interrupt bit.

Bit	Description
31:0	**Interrupt_Status_Bits** Project: All This field contains the non-persistent values of all interrupt status bits. The IMR register selects which of these interrupt conditions are reported in the persistent IIR.

Value	Name	Description	Project
0b	Condition Doesn't exist	Interrupt Condition currently does not exist	All
1b	Condition Exists	Interrupt Condition currently exists	All

Programming Notes
Some inputs to this register are short pulses; therefore software should not expect to use this register to sample these conditions.

2.1.3 IMR — Interrupt Mask

IMR	
Register Type:	MMIO
Project:	All
Default Value:	FFFFFFFFh
Access:	R/W
Size (in bits):	32

See the interrupt bit definition tables to find the source event for each interrupt bit.

Bit	Description
31:0	**Interrupt_Mask_Bits** Project: All This field contains a bit mask which selects which interrupt bits from the ISR are reported in the IIR.

Value	Name	Description	Project
0b	Not Masked	Not Masked – will be reported in the IIR	All
1b	Masked	Masked – will not be reported in the IIR	All

2.1.4 IIR — Interrupt Identity

IIR	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access:	R/W Clear
Size (in bits):	32

See the interrupt bit definition table to find the source event for each interrupt bit.

Bit	Description
31:0	**Interrupt_Identity_Bits** Project: All This field holds the persistent values of the interrupt bits from the ISR which are unmasked by the IMR. If enabled by the IER, bits set in this register will generate a PCH display interrupt. Bits set in this register will remain set (persist) until the interrupt condition is cleared by writing a '1' to the appropriate bits. For each bit, the IIR can store a second pending interrupt if two or more of the same interrupt conditions occur before the first condition is cleared. Upon clearing the interrupt, the IIR bit and PCH display interrupt will momentarily go low, then return high to indicate there is another interrupt pending. Only the rising edge of the PCH Display interrupt will cause the North Display IIR (DEIIR) PCH Display Interrupt event bit to be set, so all PCH Display Interrupts, including back to back interrupts, must be cleared here before a new PCH Display Interrupt can cause the DEIIR to be set.

Value	Name	Description	Project
0b	Condition Not Detected	Interrupt Condition Not Detected	All
1b	Condition Detected	Interrupt Condition Detected (may or may not have generated a PCH display interrupt)	All

2.1.5 IER — Interrupt Enable

IER	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

See the interrupt bit definition tables to find the source event for each interrupt bit.

Bit	Description
31:0	**Interrupt_Enable_Bits** Project: All The bits in this register enable a PCH display interrupt to be generated whenever the corresponding bit in the IIR becomes set. A disabled interrupt will still appear in the IIR register to allow polling of interrupt sources.

Value	Name	Description	Project
0b	Disable	Disable	All
1b	Enable	Enable	All

2.1.6 SHOTPLUG_CTL — South Hot Plug Control

SHOTPLUG_CTL	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

Bit	Description
31:21	**Reserved** Project: All Format:
20	**DP_D_HPD_Input_Enable** Project: All Default Value: 0b Controls the state of the HPD buffer for the digital port D. The buffer state is independent of whether the port is enabled or not.

Value	Name	Description	Project
0b	Disable	Buffer disabled	All
1b	Enable	Buffer enabled. Hot plugs bit reflect the electrical state of the HPD pin	All

SHOTPLUG_CTL

19:18 — **DP_D_HPD_Short_Pulse_Duration**

Project: All

Default Value: 0b

These bits define the duration of the pulse defined as a short pulse for the digital port D.

Value	Name	Description	Project
00b	2ms	2mS	All
01b	4.5ms	4.5mS	All
10b	6ms	6mS	All
11b	100ms	100mS	All

17:16 — **DP_D_HPD_Status**

Project: All

Default Value: 0b

This reflects hot plug detect status on the digital port D. Graphics software must write a one to these bits to clear the status. This bit is used for either monitor hotplug/unplug or for notification of a sink event. When either a long or short pulse is detected, one of these bits will set. These bits are ORed together to go to the main ISR hotplug register bit.

Value	Name	Description	Project
00b	No Detect	Digital port hot plug event not detected	All
X1b	Short Detect	Digital port short pulse hot plug event detected	All
1Xb	Long Detect	Digital port long pulse hot plug event detected	All

15:13 — **Reserved** Project: All Format:

12 — **DP_C_HPD_Input_Enable**

Project: All

Default Value: 0b

Controls the state of the HPD buffer for the digital port C. The buffer state is independent of whether the port is enabled or not.

Value	Name	Description	Project
0b	Disable	Buffer disabled	All
1b	Enable	Buffer enabled. Hot plugs bit reflect the electrical state of the HPD pin	All

SHOTPLUG_CTL

11:10

DP_C_HPD_Short_Pulse_Duration

Project: All

Default Value: 0b

These bits define the duration of the pulse defined as a short pulse for the digital port C.

Value	Name	Description	Project
00b	2ms	2mS	All
01b	4.5ms	4.5mS	All
10b	6ms	6mS	All
11b	100ms	100mS	All

9:8

DP_C_HPD_Status

Project: All

Default Value: 0b

This reflects hot plug detect status on the digital port C. Graphics software must write a one to these bits to clear the status. This bit is used for either monitor hotplug/unplug or for notification of a sink event. When either a long or short pulse is detected, one of these bits will set. These bits are ORed together to go to the main ISR hotplug register bit.

Value	Name	Description	Project
00b	No Detect	Digital port hot plug event not detected	All
X1b	Short Detect	Digital port short pulse hot plug event detected	All
1Xb	Long Detect	Digital port long pulse hot plug event detected	All

7:5

Reserved Project: All Format:

4

DP_B_HPD_Input_Enable

Project: All

Default Value: 0b

Controls the state of the HPD buffer for the digital port B. The buffer state is independent of whether the port is enabled or not.

Value	Name	Description	Project
0b	Disable	Buffer disabled	All
1b	Enable	Buffer enabled. Hot plugs bit reflect the electrical state of the HPD pin	All

SHOTPLUG_CTL	
3:2	**DP_B_HPD_Short_Pulse_Duration** Project: All Default Value: 0b These bits define the duration of the pulse defined as a short pulse for the digital port B.

Value	Name	Description	Project
00b	2ms	2mS	All
01b	4.5ms	4.5mS	All
10b	6ms	6mS	All
11b	100ms	100mS	All

1:0	**DP_B_HPD_Status** Project: All Default Value: 0b This reflects hot plug detect status on the digital port B. Graphics software must write a one to these bits to clear the status. This bit is used for either monitor hotplug/unplug or for notification of a sink event. When either a long or short pulse is detected, one of these bits will set. These bits are ORed together to go to the main ISR hotplug register bit.

Value	Name	Description	Project
00b	No Detect	Digital port hot plug event not detected	All
X1b	Short Detect	Digital port short pulse hot plug event detected	All
1Xb	Long Detect	Digital port long pulse hot plug event detected	All

2.2 GMBUS and I/O Control Registers (C5000h–C5FFFh)

2.2.1 GPIO Pin Usage (By Functions)

GPIO pins allow the support of simple query and control functions such as DDC and I^2C interface protocols. GPIO pins exist in pairs (for the most part) and provide a mechanism to control external devices through a register programming interface. GPIO pins can be set to a level or the value of the pin can be read. This allows for a “bit banging” version of an I2C interface to be implemented. An additional function of using the GMBUS engine to run the I2C protocols is also allowed. Refer to the *CSpec* for GPIO signal descriptions. Refer to the *Philips I2C-BUS SPECIFICATION version 2.1* for a description of the I2C bus and protocol.

The number and names of the GPIO pins vary from device type to device type. Some of the GPIO pins will be muxed with other functions and are only available when the other function is not being used. The following subsections describe the GPIO pin to register mapping for the various devices. OEMs have the ability to remap these functions onto other pins as long as the hardware limitations are observed.

Port	Pin Use (Name)	GMBUS Use	Internal Pullup	I^2C	Device	Description
5	HDMI/DPD CTLDATA	Yes	No (weak pulldown on reset)	Yes	All	DDC for HDMI connection via the integrated HDMI port D
	HDMI/DPD CTLCLK		No	Yes		
4	HDMIB CTLDATA	Yes	No (weak pulldown on reset)	Yes	All	Used for programming HDMI device via GMBUS protocol.
	HDMIB CTLCLK		No	Yes		
3	HDMI/DPC CTLDATA	Yes	No (weak pulldown on reset)	Yes	All	DDC for HDMI connection via the integrated HDMI port C.
	HDMI/DPC CTLCLK		No	Yes		
2	LVDS DDC Data (DDCLDATA)	Yes	No	Yes	All	DDC for Digital Display connection via the integrated LVDS
	LVDS DDC Clock (DDCLCLK)			Yes		
1	I2C Data (LCLKCTRLB)	Yes	No	Yes	All	For control of SSC clock generator devices on motherboard. Support can be optionally I2C or control level.
	I2C Clock (LCLKCTRLA)			Yes		
0	DAC DDC Data (DDCADATA)	Yes	No	Yes	All	DDC for Analog monitor (VGA) connection. This cannot be shared with other DDC or I2C pairs due to legacy monitor issues.
	DAC DDC Clock (DDCACLK)			Yes		

2.2.2 GPIO_CTL –GPIO Control

<table>
<tr><th colspan="2">GPIO Control Register Format</th></tr>
<tr><td>Project:</td><td>All</td></tr>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:13</td><td>Reserved Project: All Format: MBZ</td></tr>
<tr><td>12</td><td>GPIO_Data_In Project: All Access: Read Only
This is the value that is sampled on the GPIO_Data pin as an input.
This input is synchronized to the Core Clock domain. Because the default setting is this buffer is an input, this bit is undefined at reset.</td></tr>
<tr><td>11</td><td>GPIO_Data_Value Project: All Access: R/W
Default Value: 1b
This is the value that should be place on the GPIO Data pin as an output. This value is only written into the register if GPIO DATA MASK is also asserted. The value will appear on the pin if this data value is actually written to this register and the GPIO Data DIRECTION VALUE contains a value that will configure the pin as an output.
Default = 1. The GPIO default clock data value is programmed to ‘1’ in hardware. The hardware drives a default of ‘1’ since the I2C interface defaults to a ‘1’. (this mimics the I2C external pull-ups on the bus)</td></tr>
<tr><td>10</td><td>GPIO_Data_Mask
Project: All
Access: Write Only
Default Value: 0b
This is a mask bit to determine whether the GPIO DATA VALUE bit should be written into the register. This value is not stored and when read returns 0.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0b</td><td>No Write</td><td>Do NOT write GPIO Data Value bit</td><td>All</td></tr>
<tr><td>1b</td><td>Write</td><td>Write GPIO Data Value bit.</td><td>All</td></tr>
</table></td></tr>
<tr><td>9</td><td>GPIO_Data_Direction_Value
Project: All
Access: R/W
Default Value: 0b
This is the value that should be used to define the output enable of the GPIO Data pin. This value is only written into the register if GPIO Data DIRECTION MASK is also asserted. The value that will appear on the pin is defined by what is in the register for the GPIO DATA VALUE bit.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0b</td><td>Input</td><td>Pin is configured as an input</td><td>All</td></tr>
<tr><td>1b</td><td>Output</td><td>Pin is configured as an output</td><td>All</td></tr>
</table></td></tr>
</table>

GPIO Control Register Format

8

GPIO_Data_Direction_Mask

Project: All

Access: Write Only

Default Value: 0b

This is a mask bit to determine whether the **GPIO DIRECTION VALUE** bit should be written into the register. This value is not stored and when read always returns 0.

Value	Name	Description	Project
0b	No Write	Do NOT write GPIO Data Direction Value bit	All
1b	Write	Write GPIO Data Direction Value bit	All

7:5

Reserved Project: All Format: MBZ

4

GPIO_Clock_Data_In Project: All Access: Read Only

This is the value that is sampled on the GPIO Clock pin as an input.

This input is synchronized to the Core Clock domain. Because the default setting is this buffer is an input, this bit is undefined at reset.

3

GPIO_Clock_Data_Value Project: All Access: R/W

Default Value: 1b

This is the value that should be place on the GPIO Clk pin as an output. This value is only written into the register if **GPIO Clock DATA MASK** is also asserted. The value will appear on the pin if this data value is actually written to this register and the **GPIO Clock DIRECTION VALUE** contains a value that will configure the pin as an output.

Default = 1. The GPIO default clock data value is programmed to '1' in hardware. The hardware drives a default of '1' since the I2C interface defaults to a '1'. (this mimics the I2C external pull-ups on the bus)

2

GPIO_Clock_Data_Mask

Project: All

Access: Write Only

Default Value: 0b

This is a mask bit to determine whether the **GPIO Clock DATA VALUE** bit should be written into the register. This value is not stored and when read always returns 0.

Value	Name	Description	Project
0b	No Write	Do NOT write GPIO Clock Data Value bit	All
1b	Write	Write GPIO Clock Data Value bit	All

GPIO Control Register Format

1	**GPIO_Clock _Direction_Value** Project: All Access: R/W Default Value: 0b This is the value that should be used to define the output enable of the GPIO Clock pin. This value is only written into the register if **GPIO Clock DIRECTION MASK** is also asserted. The value that will appear on the pin is defined by what is in the register for the **GPIO Clock DATA VALUE** bit.

Value	Name	Description	Project
0b	Input	Pin is configured as an input and the output driver is set to tri-state	All
1b	Output	Pin is configured as an output	All

0	**GPIO_Clock_Direction_Mask** Project: All Access: Write Only Default Value: 0b This is a mask bit to determine whether the **GPIO Clock DIRECTION VALUE** bit should be written into the register. This value is not stored and when read returns 0.

Value	Name	Description	Project
0b	No Update	Do NOT update the GPIO Clock Direction Value bit on a write	All
1b	Update	Update the GPIO Clock Direction Value bit. on a write operation to this register	All

GPIO_CTL

Register Type: MMIO
Project: All
Default Value: 000U1000b
Access: R/W
Size (in bits): 6x32

These registers define the control of sets of the "general purpose" I/O pins. Each register controls a pair of pins that can be used for general purpose control, but most are designated for specific functions according to the requirements of the device and the system that the device is in. Each pin of the two pin pair is designated as a clock or data for descriptive purposes. See the table at the beginning of this section to determine which pins/registers are supported and their intended functions. **Board design variations are possible and would affect the usage of these pins.**

The registers that control digital display (HDMI/DVI, DisplayPort) pins should only be utilized if the Port Detected bit in the related control register is set to 1.

DWord	Bit	Description
0	31:0	GPIOCTL_0 Project: All Format: GPIO Control Register Format

GPIO_CTL						
1	31:0	GPIOCTL_1	Project:	All	Format:	GPIO Control Register Format
2	31:0	GPIOCTL_2	Project:	All	Format:	GPIO Control Register Format
3	31:0	GPIOCTL_3	Project:	All	Format:	GPIO Control Register Format
4	31:0	GPIOCTL_4	Project:	All	Format:	GPIO Control Register Format
5	31:0	GPIOCTL_5	Project:	All	Format:	GPIO Control Register Format

2.2.3 GMBUS Controller Programming Interface

The GMBUS (Graphic Management Bus) can be used to indirectly access/control devices connected to a GMBUS bus as an alternate to bit-wise programming via software.

The GMBUS interface is I^2C compatible. The basic features are listed as follow:

1. Works as the master of a single master bus.
2. The bus clock frequency is selectable by software to be 50KHz, 100KHz, 400KHz , and 1MHz
3. The GMBUS controller can be attached to the selected GPIO pin pairs.
4. 7 or 10-Bit Slave Address and 8- or 16-bit index.
5. Hardware byte counter to track the data transmissions/reception
6. Timing source from core display clock.
7. There is a double buffered data register and a 9 bit counter to support 0 byte to 256 byte transfers.
8. The slave device can cause a stall by pulling down the clock line (Slave Stall), or delay the slave acknowledge response.
9. The master controller detects and reports time out conditions for a stall from a slave device or delayed or missing slave acknowledge.
10. Interrupt may optionally be generated.
11. There is no support for ring buffer based operation of GMBUS. The GMBUS is controlled by a set of memory mapped IO registers. Status is reported through the GMBUS status register.
12. GMBUS does not directly support segment pointer addressing as defined by the Enhanced Display Data Channel standard. Segment pointer addressing for EDDC shall be supported as follows:

a. Use bit bashing (GPIO programming) to complete segment pointer register write over the target I2C port **without terminating in a stop or wait cycle**.

b. Terminate bit bashing phase with both I2C lines pulled high by tri-stating the data line before the clock line. Follow EDDC requirement for response received from slave device.

c. Initiate gmbus cycle as required to transfer EDID follow normal procedure.

The byte counter register is a read/write register, and in receiving mode, is used to track the data bytes received. There is a status register to indicate the error condition, data buffer busy, time out, and data complete acknowledgement.

2.2.3.1 GMBUS0—GMBUS Clock/Port Select

GMBUS0—GMBUS Clock/Port Select

Register Type: MMIO
Address Offset: C5100h
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 32

The GMBUS0 register will set the clock rate of the serial bus and the device the controller is connected to. The clock rate options are 50 KHz, 100 KHz, 400 KHz, and 1MHz. This register should be set before the first data valid bit is set, because it will be read only at the very first data valid bit, and not read during the period of the transmission until stop is issued and next first data valid bit is set.

Bit	Description
31:12	**Reserved** Project: All Format:
11	**Reserved**
10:8	**GMBUS_Rate_Select** Project: All Default Value: 0b These two bits select the rate that the GMBUS will run at. It also defines the AC timing parameters used. It should only be changed when between transfers when the GMBUS is idle.

Value	Name	Description	Project
000b	100KHz	100 KHz	All
001b	50KHz	50 KHz	All
010b	400KHz	400 KHz	All
1XXb	RESERVED	Reserved	All

7:3	**Reserved** Project: All Format: MBZ

GMBUS0—GMBUS Clock/Port Select

2:0

Pin_Pair_Select

Project: All

Default Value: 0b

This field selects a GMBUS pin pair for use in the GMBUS communication. Use the table above to determine which pin pairs are available for a particular device and the intended function of that pin pair. Note that it is not a straight forward mapping of port numbers to pair select numbers.

Value	Name	Description	Project
000b	None	None (disabled)	All
001b	LCTRCLK	LCTRCLKA, LCTRLCLKB SSC Clock Device	All
010b	Analog Mon	Dedicated Analog Monitor DDC Pins (DDC1DATA, DDC1CLK)	All
011b	LVDS	Integrated Digital Panel DDC Pins, LVDS	All
100b	Port C	DP/HDMI port C Use	All
110b	Port D	DP/HDMI port D Use	All
111b	Reserved	Reserved	All

2.2.3.2 GMBUS1—GMBUS Command/Status

GMBUS1—GMBUS Command/Status

Register Type: MMIO
Address Offset: C5104h
Project: All
Default Value: 00000000h
Access: R/W Protect
Size (in bits): 32

This register lets the software indicate to the GMBUS controller the slave device address, register index, and indicate when the data write is complete.

When the SW_CLR_INT bit is asserted, all writes to the GMBUS2, GMBUS3, and GMBUS4 registers are discarded. The GMBUS1 register writes to any other bit except the SW_CLR_INT are also lost. Reads to these registers always work normally regardless of the state of the SW_CLR_INT bit.

Bit	Description
31	**Software_Clear_Interrupt** Project: All Access: R/W Default Value: 0b (SW_CLR_INT) This bit must be clear for normal operation. Setting the bit then clearing it acts as local reset to the GMBUS controller. This bit is commonly used by software to clear a BUS_ERROR when a slave device delivers a NACK.

Value	Name	Description	Project
0b	Clear HW_RDY	If this bit is written as a zero when its current state is a one, will clear the **HW_RDY** bit and allows register writes to be accepted to the GMBUS registers (Write Protect Off). This bit is cleared to zero when an event causes the HW_RDY bit transition to occur.	All
1b	Assert HW_RDY	Asserted by software after servicing the GMBUS interrupt. Setting this bit causes the INT status bit to be cleared. Setting (1) this bit also asserts the **HW_RDY** bit (until this bit is written with a 0). When this bit is set, no writes to GMBUS registers will cause the contents to change with the exception of this bit which can be written.	All

Bit	Description
30	**Software_Ready** Project: All Default Value: 0b (SW_RDY) Data handshake bit used in conjunction with **HW_RDY** bit.

Value	Name	Description	Project
0b	De-Assert	De-asserted via the assertion event for **HW_RDY** bit	All
1b	SW Assert	When asserted by software, results in de-assertion of **HW_RDY** bit	All

GMBUS1—GMBUS Command/Status

29 — **Enable_Timeout**

Project: All

Default Value: 0b

(ENT) Enables timeout for slave response. When this bit is enabled and the slave device response has exceeded the timeout period, the GMBUS Slave Stall Timeout Error interrupt bit is set.

Value	Name	Description	Project
0b	Disable	Disable timeout counter	All
1b	Enable	Enable timeout counter	All

28 — **Reserved** Project: All Format:

27:25 — **Bus_Cycle_Select**

Project: All

Default Value: 0b

GMBUS cycle will always consist of a START followed by Slave Address, followed by an optional read or write data phase. A read cycle with an index will consist of a START followed by a Slave Address a WRITE indication and the INDEX and then a RESTART with a Slave Address and an optional read data phase. The GMBUS cycle will terminate either with a STOP or by entering a wait state. The WAIT state is exited by generating a STOP or by starting another GMBUS cycle.

This can only cause a STOP to be generated if a GMBUS cycle is generated, the GMBUS is currently in a data phase, or it is in a WAIT phase:

Note that the three bits can be decoded as follows:

27 = STOP generated

26 = INDEX used

25 = Cycle ends in a WAIT

Value	Name	Description	Project
000b	No cycle	No GMBUS cycle is generated	All
001b	No Index, No Stop, Wait	GMBUS cycle is generated without an INDEX, with no STOP, and ends with a WAIT	All
010b	Reserved	Reserved	All
011b	Index, No Stop, Wait	GMBUS cycle is generated with an INDEX, with no STOP, and ends with a WAIT	All
100b	Gen Stop	Generates a STOP if currently in a WAIT or after the completion of the current byte if active	All
101b	No Index, Stop	GMBUS cycle is generated without an INDEX and with a STOP	All
110b	Reserved	Reserved	All
111b	Index, Stop	GMBUS cycle is generated with an INDEX and with a STOP	All

24:16 — **Total_Byte_Count** Project: All Format:

(9-bits) This determines the total number of bytes to be transferred during the DATA phase of a GMBUS cycle. The DATA phase can be prematurely terminated by generating a STOP while in the DATA phase (see Bus Cycle Select). Do not change the value of this field during GMBUS cycles transactions.

GMBUS1—GMBUS Command/Status

15:8	**8_bit_Slave_Register_Index** Project: All Format: (INDEX) This field specifies the 8-bits of index to be used for the generated bus write transaction or the index used for the WRITE portion of the WRITE/READ pair. It only has an effect if the enable Index bit is set. Do not change this field during a GMBUS transaction.
7:0	**Slave_Address_And_Direction** Project: All Bits 7:1 = **7-bit GMBUS Slave Address (SADDR):** When a GMBUS cycle is to be generated using the Bus Cycle Select field, this field specifies the value of the slave address that is to be sent out. For use with 10-bit slave address devices, set this value to 11110XXb (where the last two bits (xx) are the two MSBs of the 10-bit address) and the slave direction bit to a write. This is followed by the first data byte being the 8 LSBs of the 10-bit slave address. Bit 0 = **Slave Direction Bit:** When a GMBUS cycle is to be generated based on the Bus Cycle Select, this bit determines if the operation will be a read or a write. A read operation with the index enabled will perform a write with just the index followed by a re-start and a read. A 1 indicates that a Read from the slave device operation is to be performed. A 0 indicates that a Write to the slave device operation is to be performed.

Value	Name	Description	Project
00000001b	General	General Call Address	All
00000000b	Start	Start Bye	All
0000001Xb	CBUS	CBUS Address	All
11110XXXb	10-bit	10-Bit addressing	All
Others	Reserved	Reserved	All

2.2.3.3 GMBUS2—GMBUS Status Register

GMBUS2—GMBUS Status Register

Register Type: MMIO
Address Offset: C5108h
Project: All
Default Value: 00000800h
Access: R/W Protect
Size (in bits): 32

Bit	Description
31:16	**Reserved** Project: All Format:
15	**INUSE** Project: All Default Value: 0b Software wishing to arbitrate for the GMBUS resource can poll this bit until it reads a zero and will then own usage of the GMBUS controller. This bit has no effect on the hardware, and is only used as semaphore among various independent software threads that don't know how to synchronize their use of this resource that may need to use the GMBUS logic. Writing a one to this bit is software's indication that the software use of this resource is now terminated and it is available for other clients.

Value	Name	Description	Project
0b	GMBUS is Acquired	Read operation that contains a zero in this bit position indicates that the GMBUS engine is now acquired and the subsequent reads of this register will now have this bit set. Writing a 0 to this bit has no effect.	All
1b	GMBUS in Use	Read operation that contains a one for this bit indicates that the GMBUS is currently allocated to someone else and "In use". Once set, a write of a 1 to this bit indicates that the software has relinquished the GMBUS resource and will reset the value of this bit to a 0.	All

Bit	Description
14	**Hardware_Wait_Phase** Project: All Access: Read Only Default Value: 0b **(HW_WAIT_PHASE)** Once in a WAIT_PHASE, the software can now choose to generate a STOP cycle or a repeated start (RESTART) cycle followed by another GMBUS transaction on the GMBUS.

Value	Name	Description	Project
0b	No Wait	The GMBUS engine is not in a wait phase.	All
1b	Wait	Set when GMBUS engine is in wait phase. Wait phase is entered at the end of the current transaction when that transaction is selected not to terminate with a STOP.	All

GMBUS2—GMBUS Status Register

13

Slave_Stall_Timeout_Error

Project: All

Access: Read Only

Default Value: 0b

This bit indicates that a slave stall timeout has occurred. It is tied to the Enable Timeout (ENT) bit.

Value	Name	Description	Project
0b	No Slave Timeout	No slave timeout has occurred	All
1b	Slave Timeout	A slave acknowledge timeout has occurred	All

12

GMBUS_Interrupt_Status

Project: All

Access: Read Only

Default Value: 0b

This bit indicates that an event that causes a GMBUS interrupt has occurred.

Value	Name	Description	Project
0b	No Interrupt	The conditions that could cause a GMBUS interrupt have not occurred or this bit has been cleared by software assertion of the **SW_CLR_INT** bit.	All
1b	Interrupt	GMBUS interrupt event occurred. This interrupt must have been one of the types enabled in the GMBUS4 register	All

GMBUS2—GMBUS Status Register

11

Hardware_Ready

Project: All

Access: Read Only

Default Value: 1b See Description Below

(HW_RDY) This provides a method of detecting when the current software client routine can proceed with the next step in a sequence of GMBUS operations. This data handshake bit is used in conjunction with the **SW_RDY** bit. When this bit is asserted by the GMBUS controller, it results in the de-assertion of the **SW_RDY** bit.

This bit resumes to normal operation when the **SW_CLR_INT** bit is written to a 0.

Value	Name	Description	Project
0b		Condition required for assertion has not occurred or when this bit is a one and: - **SW_RDY** bit has been asserted - During a GMBUS read transaction, after the each read of the data register - During a GMBUS write transaction, after each write of the data register - **SW_CLR_INT** bit has been cleared	All
1b		This bit is asserted under the following conditions: - After a reset or when the transaction is aborted by the setting of the **SW_CLR_INT** bit - When an active GMBUS cycle has terminated with a STOP - When during a GMBUS write transaction, the data register needs and can accept another four bytes of data - During a GMBUS read transaction, this bit is asserted when the data register has four bytes of new data or the read transaction DATA phase is complete and the data register contains the last few bytes of the read data	All

10

NAK_Indicator

Project: All

Access: Read Only

Default Value: 0b

Value	Name	Description	Project
0b	No bus error	No bus error has been detected or **SW_CLR_INT** has been written as a zero since the last bus error	All
1b	No Ack	Set by hardware if any expected device acknowledge is not received from the slave within the timeout	All

GMBUS2—GMBUS Status Register	
9	**GMBUS_Active** Project: All Access: Read Only Default Value: 0b **(GA)** This is a status bit that indicates whether the GMBUS controller is in an IDLE state or not.

Value	Name	Description	Project
0b	Idle	The GMBUS controller is currently IDLE	All
1b	Active	This indicates that the bus is in START, ADDRESS, INDEX, DATA, WAIT, or STOP Phase. Set when GMBUS hardware is not IDLE.	All

8:0	**Current_Byte_Count** Project: All Access: Read Only Can be used to determine the number of bytes currently transmitted/received by the GMBUS controller hardware. Set to zero at the start of a GMBUS transaction data transfer and incremented after the completion of each byte of the data phase. Note that because reads have internal storage, the byte count on a read operation may be ahead of the data that has been accepted from the data register.

2.2.3.4 GMBUS3—GMBUS Data Buffer

GMBUS3—GMBUS Data Buffer	
Register Type:	MMIO
Address Offset:	C510Ch
Project:	All
Default Value:	00000000h
Access:	R/W Protect
Size (in bits):	32
Double Buffer Update Point:	Start of next Vblank
Double Buffer Armed By:	HW_RDY

This is data read/write register. This register is double buffered. Bit 0 is the first bit sent or read, bit 7 is the 8th bit sent or read, all the way through bit 31 being the 32nd bit sent or read. For GMBUS write operations with a non-zero byte count, this register should be written with the data before the GMBUS cycle is initiated. For byte counts that are greater than four bytes, this register will be written with subsequent data only after the HW_RDY status bit is set indicating that the register is now ready for additional data. For GMBUS read operations, software should wait until the HW_RDY bit indicates that the register contains the next set of valid read data before reading this register.

Bit	Description			
31:24	**Data Byte 3**	Project:	All	Format:
23:16	**Data Byte 2**	Project:	All	Format:
15:8	**Data Byte 1**	Project:	All	Format:
7:0	**Data Byte 0**	Project:	All	Format:

2.2.3.5 GMBUS4—GMBUS Interrupt Mask

GMBUS4—GMBUS Interrupt Mask

Register Type: MMIO
Address Offset: C5110h
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 32

Bit	Description
31:5	**Reserved** Project: All Format:
4:0	**Interrupt_Mask** Project: All Default Value: 0b This field specifies which GMBUS interrupts events may contribute to the setting of GMBUS interrupt status bit in second level interrupt status register.

Value	Name	Description	Project
0XXXXb	GMBUS Slave stall TO Disable	Disable GMBUS Slave stall timeout interrupt	All
1XXXXb	GMBUS Slave stall TO Enable	Enable GMBUS Slave stall timeout interrupt	All
X0XXXb	GMBUS NAK Disable	Disable GMBUS NAK interrupt	All
X1XXXb	GMBUS NAK Enable	Enable GMBUS NAK interrupt	All
XX0XXb	GMBUS Idle Disable	Disable GMBUS Idle interrupt	All
XX1XXb	GMBUS Idle Enable	Enable GMBUS Idle interrupt	All
XXX0Xb	HW Wait Disable	Disable Hardware wait (GMBUS cycle without a stop has completed) Interrupt	All
XXX1Xb	HW Wait Enable	Enable Hardware wait (GMBUS cycle without a stop has completed) Interrupt	All
XXXX0b	HW Ready Disable	Disable Hardware ready (Data has been transferred) interrupt	All
XXXX1b	HW Ready Enable	Enable Hardware ready (Data has been transferred) interrupt	

2.2.3.6 GMBUS5—GMBUS 2 Byte Index Register

GMBUS5—GMBUS 2 Byte Index Register	
Register Type:	MMIO
Address Offset:	C5120h
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

This register provides a method for the software indicate to the GMBUS controller the 2 byte device index.

Bit	Description
31	**2_Byte_Index_Enable** Project: All Format: When this bit is asserted (1), then bits 15:0 are used as the index. Bits 15:8 are used in the first byte which is the most significant index bits. The slave index in the GMBUS1<15:8> are ignored. Bits 7:0 are used in the second byte which is the least significant index bits.
30:16	**Reserved** Project: All Format:
15:0	**2_Byte_Slave_Index** Project: All Format: This is the 2 byte index used in all GMBUS accesses when bit 31 is asserted (1).

2.3 Display Clock Control Registers (C6000h–C6FFFH)

Pixel Data Rate	Dot Clock	Dual Channel?	External Clock	Data Clock Rate	Multiplier
HDMI 25-50MHz	100-200MHz	NO	100-200MHz	1.0-2.0GHz	4x
HDMI 50-100MHz	100-200MHz	NO	100-200MHz	1.0-2.0GHz	2x
HDMI 100-225MHz	100-225MHz	NO	100-225MHz	1.0-2.25GHz	1x
LVDS 25-112MHz	25-112MHz	NO	25-112MHz	175-784MHz	1x
LVDS 80-224MHz	80-224MHz	YES	80-224MHz	280-784MHz	1x

Display Modes	Display Clock Frequency Range (MHz)
CRT DAC	25-350
HDMI	25-225 (pixel rate can differ from clock frequency)
LVDS (Single Channel)	25-112

Display Modes	Display Clock Frequency Range (MHz)
LVDS (Dual Channel)	80-224
Display Port	162, 270 (pixel rate can differ from clock frequency)

The PLL frequency selection must be done such that the internal VCO frequency is within its limits. The PLL Frequency is based on the selected register and the following formula. The post divider register value limits are different for LVDS mode.

Reference Frequency: 120MHz for CRT, HDMI, LVDS. 100MHz for the FDI.

DotClk_Frequency = (ReferenceFrequency * (5* (M1+2)+(M2+2)) / (N+2))/ (P1* P2)

Item	Units	Range	Notes
Dot Clock	Frequency	20-350	MHz (Combining ALL modes)
VCO	Frequency	1760-3510	MHz
N – Counter	Value	3-8	
M – Counter	Value	79-127	M=5*(M1+2)+(M2+2)
M1 and M2		M1 > M2	
M1	Value	12-22	
M2	Value	5-9	
P-Div	Value	5-80	Combined P1 and P2 for DAC mode
P-Div	Value	28-112	Combined P1 and P2 for LVDS mode
P1-Div	Value	1-8	All modes

Note: For HDMI 12bpc usage model, the PCH display pixel clock should be programmed at 1.5x the effective pixel clock of the CPU display. This needs to be taken into account when setting the post divisors.

2.3.1 DPLL_CTL—DPLL Control

DPLL_CTL

Register Type: MMIO
Project: All
Default Value: 04800080h
Access: R/W Protect
Size (in bits): 32
Double Buffer Update Point: Transcoder vertical blank, except as stated
Write Protect by Panel Power Sequencer when panel is connected to this transcoder.

Bit	Description
31	**DPLL_VCO_Enable** Project: All Access: R/W Default Value: 0b This bit will enable or disable the PLL VCO. Disabling the PLL will cause the display clock to stop.

Value	Name	Description	Project
0b	Disable	DPLL is disabled in its lowest power state	All
1b	Enable	DPLL is enabled and operational	All

Bit	Description
30	**DPLL_High_Speed_IO_clock_En** Project: All Default Value: 0b

Value	Name	Description	Project
0b	Disable	High Speed IO Clock Disabled	All
1b	Enable	High Speed IO Clock Enabled (must be set in HDMI, and DisplayPort modes)	All

Bit	Description
29:28	**Reserved** Project: All Format: MBZ
27:26	**DPLL_Mode_Select** Project: All Default Value: 01b Non-LVDS Configure the DPLL for various supported Display Modes

Value	Name	Description	Project
00b	Reserved	Reserved	All
01b	Non-LVDS	DPLL in DAC/HDMI/DisplayPort mode (default)	All
10b	LVDS	DPLL in LVDS mode	All
11b	Reserved	Reserved	All

DPLL_CTL

25:24

FP0_FP1_P2_Clock_Divide

Project: All

Default Value: 00b

Value	Name	Description	Project
00b	Div 14_10	Divide by 14 for Single-Channel LVDS Divide by 10 for DisplayPort. Also used for DAC and HDMI modes with Dot Clock <= 225MHz	All
01b	Div 7_5	Divide by 7 for Dual-Channel LVDS Divide by 5 for DAC and HDMI modes with Dot Clock > 225MHz	All
Others	Reserved	Reserved	All

23:16

FP0_P1_Post_Divisor

Project: All

Default Value: 80h Divide by eight

Writes to this byte finalize the write of m, n and p values into the PLL when the PLL is disabled. Writing to FP1 when FP0 is in use (or vice versa) is also allowed. **Writes to this register take effect immediately.**

Value	Name	Description	Project
00000001b	1	Divide by one	All
00000010b	2	Divide by two	All
00000100b	3	Divide by three	All
00001000b	4	Divide by four	All
00010000b	5	Divide by five	All
00100000b	6	Divide by six	All
01000000b	7	Divide by seven	All
10000000b	8	Divide by eight (default)	All
Others	Reserved	Reserved	All

DPLL_CTL

15:13 **PLL_Reference_Input_Select**

Project: All

Default Value: 000b

(Not Double Buffered) The PLL reference should be selected based on the display device that is being driven. The standard reference clock is used for CRT modes using the analog port CRT DAC or LCD panels for the integrated LVDS.

Value	Name	Description	Project
000b	DREFCLK	DREFCLK (default is 120 MHz) for DAC/HDMI/DVI/DisplayPort	All
001b	Super SSC	120MHz super-spread clock	All
011b	SSC	Spread spectrum input clock (120MHz default) for LVDS/DisplayPort	All
Others	Reserved	Reserved	All

12 **Reserved** Project: All Format:

11:9 **DPLL_ HDMI_multiplier**

Project: All

Default Value: 000b (1X)

Range 0..3

In order to keep the clock rate to a more narrow range of rates, the multiplier is set and the Display PLL programmed to a multiple of the display mode's actual clock rate. The DPLL must be enabled and stable before setting these bits. These bits must be programmed after DPLL_SEL is programmed

Value in this register = multiplication factor - 1

8 **Reserved** Project: All Format: MBZ

DPLL_CTL	
7:0	**FP1_P1_Post_Divisor** Project: All Default Value: 80h Divide by eight Writes to this byte finalize the write of m, n and p values into the PLL when the PLL is disabled. Writing to FP1 when FP0 is in use (or vice versa) is also allowed. Writes to this register take effect immediately.

Value	Name	Description	Project
00000001b	1	Divide by one	All
00000010b	2	Divide by two	All
00000100b	3	Divide by three	All
00001000b	4	Divide by four	All
00010000b	5	Divide by five	All
00100000b	6	Divide by six	All
01000000b	7	Divide by seven	All
10000000b	8	Divide by eight (Default)	All
Others	Reserved	Reserved	All

2.3.2 DPLL_FP0—DPLL Divisor 0

DPLL_FP0

Register Type: MMIO
Project: All
Default Value: 00030D07h
Access: R/W Protect
Size (in bits): 32
Double Buffer Update Point: Transcoder vertical blank
Write Protect by Panel Power Sequencer when panel is connected to this transcoder.

Bit	Description
31:28	**Reserved** Project: All Format: MBZ
27	**Frequency_doubler_clock_enable** Project: All Default Value: 0b This bit enables/disables the frequency doubler clock. When the VCO clock to the doubler is disabled, the circuit does not dissipate power and its output clock is not available

Value	Name	Description	Project
0b	Disable	Disables clock of frequency doubler	All
1b	Enable	Enables clock of frequency doubler	All

DPLL_FP0

26:25	**Reserved** Project: All Format: MBZ
24:22	**CB_Tuning** Project: All Default Value: 000b These bits are used for CB tuning the Display PLL Analog core on PCH. These bits are required to improve the jitter performance and VCO headroom of the Display PLL across Process, Voltage and Temperature variations. The CB tune should be turned on when the M/N ratio is less than a certain value given in the table below. The bits should be programmed to 0x011 to turn the complete CB cap on.

Display Mode	If M/N Ratio is less than	Bits <24:22>
DAC	21.00	011
HDMI	21.00	011
LVDS 1ch (120mhz input clock)	21.00	011
LVDS 2ch (120mhz input clock)	25.00	011
LVDS 1ch (100mhz input clock)	25.00	011
LVDS 2ch (100mhz input clock)	25.00	011

Example1:

In DAC mode, for pixel clock = 31MHz, N=4; M=83; P=80;

Therefore M/N ratio = 20.75 which is < 21.00 mentioned in the table above. Hence the CB tune bits <24:22> need to be programmed to 011.

Example2:

In DAC mode, for pixel clock = 31.5MHz, N=4; M=84; P=80;

Therefore M/N ratio = 21 which equal to the value of M/N ration mentioned in the table above. Hence the CB tune bits <24:22> need to be programmed to 000.

Value	Name	Description	Project
000b	Off	CB Tune Off	All
011b	100%	CB Tune 100% On	All

21:16	**FP0_N_Divisor** Project: All Format: N-Divisor value calculated for the desired output frequency. The register value is programmed two less than the actual divisor.
15:14	**Reserved** Project: All Format: MBZ

DPLL_FP0	
13:8	**FP0_M1_Divisor** Project: All Format: M-Divisor value calculated for the desired output frequency. The register value is programmed to two less than the actual divisor.
7:6	**Reserved** Project: All Format: MBZ
5:0	**FP0_M2_Divisor** Project: All Format: M-Divisor value calculated for the desired output frequency. The register value is programmed two less than the actual divisor.

2.3.3 DPLL_FP1—DPLL Divisor 1

DPLL_FP1	
Register Type:	MMIO
Project:	All
Default Value:	00030D07h
Access:	R/W Protect
Size (in bits):	32
Double Buffer Update Point:	Transcoder vertical blank
Write Protect by Panel Power Sequencer when panel is connected to this transcoder.	
Bit	**Description**
31:25	**Reserved** Project: All Format: MBZ
24:22	**CB_Tuning** Project: All Default Value: 000b See FP0 CB_Tuning description

Value	Name	Description	Project
000b	Off	CB Tune Off	All
011b	100%	CB Tune 100% On	All

21:16	**FP1_N_Divisor** Project: All Format: N-Divisor value calculated for the desired output frequency. The register value is programmed two less than the actual divisor.
15:14	**Reserved** Project: All Format: MBZ
13:8	**FP1_M1_Divisor** Project: All Format: M-Divisor value calculated for the desired output frequency. The register value is programmed to two less than the actual divisor.
7:6	**Reserved** Project: All Format: MBZ

DPLL_FP1	
5:0	**FP1_M2_Divisor** Project: All Format: M-Divisor value calculated for the desired output frequency. The register value is programmed two less than the actual divisor.

2.3.4 DREF_CONTROL — Display Reference Clock Control Register

DREF_CONTROL — Display Reference Clock Control Register

Register Type: MMIO
Address Offset: C6200h
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 32

Bit	Description
31:15	**Reserved** Project: All Format: MBZ
14:13	**120MHz_CPU_source_output_enable** Project: All Default Value: 00b

Value	Name	Description	Project
00b	Disabled	Source output to CPU disabled	All
01b	Reserved	Rerserved	All
10b	Downspread	-0.5% SSC downspread source output to CPU enabled. Both the 120MHz SSC source (bits 12:11) and the SSC1 modulator (bit1) must be enabled prior to enabling this output	All
11b	Non-spread	Non-spread source output to CPU enabled. The 120MHz non-SSC source (bit 10:9) must be enabled prior to enabling this output	All

DREF_CONTROL — Display Reference Clock Control Register

12:11

120MHz_SSC_source_en

Project: All

Default Value: 00b

This bit enables the 120MHz SSC source used as a reference for DisplayPort or CPU

Value	Name	Description	Project
00b	Disabled	Source disabled	All
01b	Reserved	Reserved for CK505 buffered source enabled	All
10b	Enabled	Integrated source enabled	All
11b	Reserved	Reserved	All

10:9

120MHz_nonspread_source_en

Project: All

Default Value: 00b

This field enables the 120MHz non-SSC source for display

Value	Name	Description	Project
00b	Disabled	Source disabled	All
01b	CK505	CK505 buffered source enabled.	All
10b	Integrated	Integrated source enabled	All
11b	Reserved	Reserved	All

8:7

120MHz_superspread_source_en

Project: All

Default Value: 00b

This field enables the 120MHz super-SSC source for display

Value	Name	Description	Project
00b	Disabled	Source disabled	All
01b	Reserved	Reserved	All
10b	Enabled	Integrated source enabled	All
11b	Reserved	Reserved	All

DREF_CONTROL — Display Reference Clock Control Register

6 — **SSC4_Spread_Mode**

Project: All

Default Value: 0b

This is the reference clock used for super-spread on LVDS. Please note that this reference is shared with SATA. If it is used for SATA it must not be used for LVDs. This bit must not be changed after bit 0 is set. It may be updated simultaneously with the update of bit 0.

Value	Name	Description	Project
0b	Downspread	Center vs downspread: this bit sets down spread on the SSC4 modulator used for superspread.	All
1b	Centerspread	Center vs downspread: this bit sets center spread on the SSC4 modulator used for superspread.	All

5:2 — **Reserved** Project: All Format: MBZ

1 — **120MHz_SSC1(-0.5%)modulation_en**

Project: All

Default Value: 0b

This bit enables the -0.5% modulator used for the 120MHz SSC source used for the CPU DisplayPort or as the -0.5% input to the DPLL in the PCH. It must be set 0uS or more after the 120MHz SSC output is enabled (this bit and bits 12:11 can be written to enable at the same time). PLL's using this clock as an input must be enabled 1uS or more after this bit is enabled to ensure a stable input.

Value	Name	Description	Project
0b	Disabled	SSC1 disabled	All
1b	Enabled	SSC1 enabled	All

0 — **120MHz_SSC4_modulation_en**

Project: All

Default Value: 0b

This bit enables the variable % modulator used for the 120MHz SSC source used for LVDS. It must be set 0uS or more after the 120MHz super-spread source is enabled (this bit and bits 8:7 can be written to enable at the same time). PLL's using this clock as an input must be enabled 1uS or more after this bit is enabled to ensure a stable input.

Value	Name	Description	Project
0b	Disabled	SSC4 disabled	All
1b	Enabled	SSC4 enabled	All

2.3.5 RAWCLK_FREQ—Rawclk Frequency

RAWCLK_FREQ—Rawclk Frequency	
Register Type:	MMIO
Address Offset:	C6204h
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

Bit	Description
31:10	**Reserved** Project: All Format:
9:0	**Rawclk_frequency** Project: All Format: Program this field to match the rawclk frequency. This is used to generate a divided down clock for miscellaneous timers in display. [DevCPT] rawclk = 125mhz

2.3.6 SSC4PARMS – SSC4 Parameters

SSC4PARMS – SSC4 Parameters	
Register Type:	MMIO
Address Offset:	C6210h
Project:	All
Default Value:	01204860h
Access:	R/W
Size (in bits):	32

Notes:

- This register must not be changed after bit 0 of register C6200h is set.
- Default values of this register are meaningless.
- 0% spread option for SSC4 should be configured by clearing bit 0 of C6200h to disable SSC4 module. In this case, registers SSC4PARMS and SSC4AuxPARMS settings have no effect and are don't care to the hardware. Bits 8:7 of C6200h still need to be configured to enable the divisor DIV4, i.e. the source of the clock.
- This register needs to be configured by the display driver for desired spread percentages. Recommended settings for 0.5% down spread and 0.5%, 1.0%, 1.5%, 2.0%, and 2.5% center spread are listed in the following look-up table. Recommended settings for half-step clock bending usage is also listed.

Register fields	half-step clock bend	0.5% down spread	0.5% center spread	1.0% center spread	1.5% center spread	2.0% center spread	2.5% center spread
6/5/4Ni_1NjRpt, [29:28]	00b	01b	01b	01b	01b	01b	01b
3Ni_1NjRpt, [26:24]	000b	010b	010b	010b	010b	010b	010b
2Ni1NjRpt,	000b	111b	111b	111b	111b	111b	111b

SSC4PARMS – SSC4 Parameters

[22:20]							
1Ni1NjRpt, [18:10]	0_0000_0000b	0_0010_1001b	0_0010_1001b	0_0000_0010b	0_0000_1010b	0_0000_0010b	0_0000_1010b
MxPhsStp, [9:3]	000_0000b	000_0101b	000_0010b	000_0101b	000_0100b	000_0101b	000_0100b
PhsIncVal, [2:0]	000b	000b	000b	000b	001b	001b	010b

Bit	Description
31:30	**Reserved** Project: All Format: MBZ
29:28	**SSC4 6/5/4Ni_1Nj Repeat Count (SSC4_6/5/4Ni_1NjRpt)** Project: All Format: U2+1 Default Value: 00b 1 times Select the number of repeats for the portion of 4, or 5, or 6 clocks of Ni and 1 clock of Nj within a dithering pattern of a step. See 1Ni_1Nj Repeat Count for more information.
27	**Reserved** Project: All Format: MBZ
26:24	**SSC4 3Ni_1Nj Repeat Count (SSC4_3Ni_1NjRpt)** Project: All Format: U3+1 Default Value: 001b 2 times Select the number of repeats for the portion of 3 clocks of Ni and 1 clock of Nj within a dithering pattern of a step. See 1Ni_1Nj Repeat Count for more information.
23	**Reserved** Project: All Format: MBZ
22:20	**SSC4 2Ni_1Nj Repeat Count (SSC4_2Ni1NjRpt)** Project: All Format: U3+1 Default Value: 010b 3 times Select the number of repeats for the portion of 2 clocks of Ni and 1 clock of Nj within a dithering pattern of a step. See 1Ni_1Nj Repeat Count for more information.
19	**Reserved** Project: All Format: MBZ
18:10	**SSC4 1Ni_1Nj Repeat Count (SSC4_1Ni1NjRpt)** Project: All Format: U9+1 Default Value: 000010010b 19 times Select the number of repeats for the portion of 1 clock of Ni and 1 clock of Nj within a dithering pattern of a step. Together, 1Ni_1Nj Repeat Count, 2Ni_1Nj Repeat Count, 3Ni_1Nj Repeat Count, and 6/5/4Ni_1Nj Repeat Count tune the width of the target 32Khz modulated period. For down spread, the target these fields are to be tuned to is one half of the modulated period. For center spread, the target is one quarter of the modulated period.
9:3	**SSC4 Max Phase Step (SSC4_MxPhsStp)** Project: All Format: U7+1 Default Value: 0001100b 13 steps Select the number of steps to reach peak of modulated phase adjustment value. Together, Max Phase Step field and Phase Increment Value field control the magnitude of the spread.

SSC4PARMS – SSC4 Parameters	
2:0	**SSC4 Phase Increment Value (SSC4_PhsIncVal)** Project: All Format: U3+1 Default Value: 000b 1 PI change per step Select the granularity of each phase step. Together, Max Phase Step field and Phase Increment Value field control the magnitude of the spread.

2.3.7 DPLL_SEL— DPLL Select

DPLL_SEL	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

Individual bits are write protected by panel power sequencing when the panel is connected to the transcoder associated with that bit.

Bit	Description
31:12	**Reserved** Project: All Format:
11	**Transcoder_C_DPLL_Enable** Project: All Default Value: 0b

Value	Name	Description	Project
0b	Disable	Disable DPLL to this transcoder	All
1b	Enable	Enable DPLL to this transcoder	All

10:9	**Reserved** Project: All Format:
8	**Transcoder_C_DPLL_Select** Project: All Default Value: 0b

Value	Name	Description	Project
0b	DPLLA	Select DPLLA for this transcoder	All
1b	DPLLB	Select DPLLB for this transcoder	All

DPLL_SEL

7	**Transcoder_B_DPLL_Enable** Project: All Default Value: 0b

Value	Name	Description	Project
0b	Disable	Disable DPLL to this transcoder	All
1b	Enable	Enable DPLL to this transcoder	All

6:5	**Reserved** Project: All Format:
4	**Transcoder_B_DPLL_Select** Project: All Default Value: 0b

Value	Name	Description	Project
0b	DPLLA	Select DPLLA for this transcoder	All
1b	DPLLB	Select DPLLB for this transcoder	All

3	**Transcoder_A_DPLL_Enable** Project: All Default Value: 0b

Value	Name	Description	Project
0b	Disable	Disable DPLL to this transcoder	All
1b	Enable	Enable DPLL to this transcoder	All

2:1	**Reserved** Project: All Format:
0	**Transcoder_A_DPLL_Select** Project: All Default Value: 0b

Value	Name	Description	Project
0b	DPLLA	Select DPLLA for this transcoder	All
1b	DPLLB	Select DPLLB for this transcoder	All

2.4 Panel Power Sequencing Registers

2.4.1 PP_STATUS—Panel Power Status Register

PP_STATUS

Register Type: MMIO
Project: All
Default Value: 08000000h
Access: Read Only
Size (in bits): 32

Bit	Description
31	**Panel_Power_On_Status** Project: All Default Value: 0b Software is responsible for enabling the embedded panel display by writing a "1" to the port enable bit only after all pipe and transcoder timing and DPLL registers are properly programmed and the PLL has locked to the reference signal. This bit is cleared to "0" only after the panel power down sequencing is completed.
30	**Require_Asset_Status** Project: All Default Value: 0b This bit indicates the status of programming of the DPLL and the selected port. A power on cycle will not be allowed unless this status indicates that the required assets are programmed and ready for use. The following conditions determine that the assets are ready: 1) Display pipe or transcoder PLL enabled and frequency locked. 2) Display pipe or transcoder enabled. 3) Port attached to the panel is enabled.

Value	Name	Description	Project
0b	Off	Indicates that the panel power down sequencing has completed. A power cycle delay may be currently active. It is safe and allowed to program timing, port, and DPLL registers. If this bit is not a zero, it activates the register write protect and writes to those registers will be ignored unless the write protect key value is set in the panel sequencing control register.	All
1b	On	In conjunction with bits Power Sequence Progress field and Power Cycle Delay Active, this bit set to a one indicates that the panel is currently powered up or is currently in the power down sequence and it is unsafe to change the timing, port, and DPLL registers for the pipe or transcoder that is assigned to the panel output.	All

Value	Name	Description	Project
0b	Not Ready	All required assets are not properly programmed	All
1b	Ready	All required assets are ready for the driving of a panel	All

PP_STATUS

29:28 — **Power_Sequence_Progress**

Project: All

Default Value: 0b

Value	Name	Description	Project
00b	None	Indicates that the panel is not in a power sequence	All
01b	Power Up	Indicates that the panel is in a power up sequence (may include power cycle delay)	All
10b	Power Down	Indicates that the panel is in a power down sequence	All
11b	Reserved	Reserved	All

27 — **Power_Cycle_Delay_Active**

Project: All

Default Value: 1b A power cycle delay (T4) is currently active

Power cycle delays occur after a panel power down sequence or after a hardware reset. On reset, a power cycle delay will occur using the default value for the timing.

Value	Name	Description	Project
0b	Not Active	A power cycle delay is not currently active	All
1b	Active	A power cycle delay (T4) is currently active	All

26:4 — **Reserved** Project: All Format:

3:0 — **Reserved**

Project: All

2.4.2 PP_CONTROL—Panel Power Control Register

PP_CONTROL—Panel Power Control Register	
Register Type:	MMIO
Address Offset:	C7204h
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

Bit	Description
31:16	**Write_Protect_Key** Project: All Default Value: 0b This field can be programmed with the key value "ABCD" to uncoditionally disable write protect. This field must be programmed with the value 0xABCD when using panel power sequencing on DisplayPort A (PP_ON_DELAYS bits 31:30 Panel_Control_port_select is set to 01b for DisplayPort A).
15:4	**Reserved** Project: All Format:
3	**eDP_VDD_Override_for_AUX** Project: All Default Value: 0b This bit is used to force on VDD for the embedded DP panel so AUX transactions can occur without enabling the panel power sequence. This is intended for panels that require VDD to be asserted before accessing AUX port on the receiver. When software clears this bit from ‘1’ to ‘0’ (disable VDD override) it must ensure that T4 power cycle delay is met before setting this bit to ‘1’ again.

Value	Name	Description	Project
0b	Not Force	VDD controlled by Panel Power Sequence state machine	All
1b	Force	Force VDD on to allow AUX transaction. Panel power sequence flow should be used regardless of the state of this bit when it is desired to enable the embedded DP main link.	All

PP_CONTROL—Panel Power Control Register

2

Backlight_Enable

Project: All

Default Value: 0b

Enabling this bit enables the panel backlight if the embedded panel is DisplayPort, as indicated in bits 31:30 of the panel power on sequencing. Software must enable this bit after training the link, and disable it when disabling the panel power state target.

Value	Name	Description	Project
0b	Disable	Backlight disabled	All
1b	Enable	Backlight enabled	All

1

Power_Down_on_Reset

Project: All

Default Value: 0b

Enabling this bit causes the panel to power down on reset warning. When system reset is initiated, the panel power down sequence begins automatically. If the panel is not on during a reset event, this bit is ignored.

This bit must be cleared to 0b prior to initiating a panel power up or power down sequence, then this bit must be restored after the sequence completes.

Value	Name	Description	Project
0b	Do not Run	Do not run panel power down sequence when reset is detected	All
1b	Run	Run panel power down sequence when system is reset	All

0

Power_State_Target

Project: All

Default Value: 0b

Writing this bit can occur any time, it will only be used at the completion of any current power cycle.

Value	Name	Description	Project
0b	Off	The panel power state target is off, if the panel is either on or in a power on sequence, a power off sequence is started as soon as the panel reaches the power on state. This may include a power cycle delay. If the panel is currently off, there is no change of the power state or sequencing done.	All
1b	On	The panel power state target is on, if the panel is in either the off state or a power off sequence, if all pre-conditions are met, a power on sequence is started as soon as the panel reaches the power off state. This may include a power cycle delay. If the panel is currently off, there is no change of the power state or sequencing done. While the panel is on or in a power on sequence, the register write lock will be enabled.	All

2.4.3 PP_ON_DELAYS—Panel Power On Sequencing Delays

PP_ON_DELAYS—Panel Power On Sequencing Delays

Register Type: MMIO
Address Offset: C7208h
Project: All
Default Value: 00000000h
Access: R/W Protect
Size (in bits): 32

Write Protect by Panel Power Sequencer

Bit	Description
31:30	**Panel_Control_port_select** Project: All Default Value: 0b These bits define to which port the embedded panel is connected. This is used for automatic control of the panel power. If the selected port is disabled then, the power sequence will not allow a panel power up.

Value	Name	Description	Project
00b	LVDS	Panel is connected to the LVDS display port	All
01b	DPA	Panel is connected to the DisplayPort A PP_CONTROL 0xC7204 bits 31:16 Write_Protect_Key must be programmed to 0xABCD when using panel power sequencing on DisplayPort A.	All
10b	DPC	Panel is connected to the DisplayPort C	All
11b	DPD	Panel is connected to the DisplayPort D	All

Bit	Description
29	**Reserved** Project: All Format:
28:16	**Power_up_delay** Project: All Format: Programmable value of panel power sequencing delay during panel power up. This provides the time delay for the T1+T2 time sequence. The time unit used is the 100us timer.
15:13	**Reserved** Project: All Format:
12:0	**Power_on_to_Backlight_enable_delay** Project: All Format: Programmable value of panel power sequencing delay during panel power up. This provides the time delay for the T5 time sequence. The time unit used is the 100us timer.

2.4.4 PP_OFF_DELAYS—Panel Power Off Sequencing Delays

PP_OFF_DELAYS—Panel Power Off Sequencing Delays	
Register Type:	MMIO
Address Offset:	C720Ch
Project:	All
Default Value:	00000000h
Access:	R/W Protect
Size (in bits):	32
Write Protect by Panel Power Sequencer	

Bit	Description
31:29	**Reserved** Project: All Format:
28:16	**Power_Down_delay** Project: All Format: Programmable value of panel power sequencing delay during power up. This provides the time delay for the T3 time sequence. The time unit used is the 100us timer.
15:13	**Reserved** Project: All Format:
12:0	**Backlight_off_to_power_down** Project: All Format: Power backlight off to power down delay. Programmable value of panel power sequencing delay during power down. This provides the time delay for the Tx time sequence. The time unit used is the 100us timer.

2.4.5 PP_DIVISOR—Panel Power Cycle Delay and Reference Divisor

PP_DIVISOR—Panel Power Cycle Delay and Reference Divisor	
Register Type:	MMIO
Address Offset:	C7210h
Project:	All
Default Value:	00186904h
Access:	R/W Protect
Size (in bits):	32

Write Protect by Panel Power Sequencer

This register selects the reference divisor and controls how long the panel must remain in a power off condition once powered down. This has a default value that allows a timer to initiate directly after device reset. If the panel limits how fast we may sequence from up to down to up again. Typically this is 0.5-1.5 seconds, but limited to 400ms in the SPWG specification. This register forces the panel to stay off for a programmed duration. Special care is needed around reset and D3 cold situations to conform to power cycle delay specifications.

Bit	Description
31:8	**Reference_divider** Project: All Default Value: 001869h 125MHz raw clock. This field provides the value of the divider used for the creation of the panel timer reference clock. The output of the divider is used as the fastest of the three time bases (100us) for all other timers. The other time bases are divided from this frequency. The value of zero should not be used. When it is desired to divide by N, the actual value to be programmed is (N/2)-1. The value should be (100 * Ref clock frequency in MHz / 2) - 1. The default value is for the 125MHz raw clock. **Example:** **Reference Clock Frequency** **Value of Field** 233MHz 2D81h 200MHz 270Fh 125MHz 1869h
7:5	**Reserved** Project: All Format:
4:0	**Power_Cycle_Delay** Project: All Format: Programmable value of time panel must remain in a powered down state after powering down. For devices coming out of reset, the default values will define how much time must pass before a power on sequence can be started. This field uses the 0.1 S time base unit from the divider. If the panel power on sequence is attempted during this delay, the power on sequence will commence once the power cycle delay is complete. Writing a value of 0 selects no delay or is used to abort the delay if it is active. During the initial power up reset, a D3 cold power cycle, or a user instigated system reset, the timer will be set to the default value and the count down will begin after the de-assertion of reset. Writing this field to a zero while the count is active will abort this portion of the sequence. This corresponds to the T4 of the SPWG specification. Note: Even if the panel is not enabled, the T4 count happens after reset. This register needs to be programmed to a “+1” value. For instance for meeting the SPWG specification of 400mS, program 5 to achieve at least 400mS delay prior to powerup.

2.5 Backlight Control Registers

2.5.1 Backlight PWM PCH Control Register 1

Backlight PWM PCH Control Register 1

Register Type: MMIO
Address Offset: C8250h
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 32

Bit	Description
31	**PWM_PCH_Enable** Project: All Default Value: 0b This bit enables the PWM counter logic in the PCH.

Value	Name	Description	Project
0b	Disable	PCH PWM disabled (drives 0 always)	All
1b	Enable	PCH PWM enabled	All

Bit	Description
30	**PWM_PCH_Override_Enable** Project: All Default Value: 0b This bit enables PWM messages from CPU to PCH to be overriden by the Backlight PWM PCH Control Register Backlight Duty Cycle Override value.

Value	Name	Description	Project
0b	Disable	Override disabled (CPU messages control PWM duty cycle)	All
1b	Enable	Override enabled (Override register value controls PWM duty cycle)	All

Bit	Description
29	**Backlight_Polarity** Project: All Default Value: 0b This field controls the polarity of the PWM signal.

Value	Name	Description	Project
0b	High	Active High	All
1b	Low	Active Low	All

Bit	Description
28	**Reserved** Project: All Format:
27:0	**Reserved** Project: All Format:

2.5.2 SBLC_BLM_CTL2—South BLM Control 2

SBLC_PWM_CTL2	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Bit	**Description**
31:16	**Backlight_Modulation_Frequency** Project: All Format: This field determines the number of time base events in total for a complete cycle of modulated backlight control. This field is normally set once during initialization based on the frequency of the clock that is being used and the desired PWM frequency. This value represents the period of the PWM stream in PCH display raw clocks multiplied by 128.
15:0	**Backlight_Duty_Cycle_Override** Project: All This value overrides the CPU control of PWM duty cycle when the PWM PCH Override Enable bit is set. This field determines the number of time base events for the active portion of the PWM backlight control. This should never be larger than the frequency field. A value of zero will turn the backlight off. A value equal to the backlight modulation frequency field will be full on. This field gets updated when it is desired to change the intensity of the backlight, it will take affect at the end of the current PWM cycle. This value represents the active time of the PWM stream in PCH display raw clock periods multiplied by 128.

3. South Display Engine Transcoder and Port Controls

3.1 Transcoder Timing

3.1.1 HTOTAL—Horizontal Total

HTOTAL	
Register Type: MMIO **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 32 Write Protect by Panel Power Sequencer when panel is connected to this transcoder.	
Bit	**Description**
31:29	**Reserved** Project: All Format: MBZ
28:16	**Horizontal_Total** Project: All This field specifies Horizontal Total size. This should be equal to the sum of the horizontal active and the horizontal blank sizes. This field is programmed to the number of pixels desired minus one. The number of pixels (before the minus one) needs to be a multiple of two when driving the LVDS port in two channel mode. This register must always be programmed to the same value as the Horizontal Blank End.
15:12	**Reserved** Project: All Format: MBZ
11:0	**Horizontal_Active** Project: All This field specifies Horizontal Active Display size. Note that the first horizontal active display pixel is considered pixel number 0. This field is programmed to the number of pixels desired minus one. The number of pixels (before the minus one) needs to be a multiple of two when driving the LVDS port in two channel mode. The minimum horizontal active display size is 64 pixels. This register must always be programmed to the same value as the Horizontal Blank Start.

3.1.2 HBLANK—Horizontal Blank

HBLANK	
Register Type: MMIO **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 32 Write Protect by Panel Power Sequencer when panel is connected to this transcoder.	
Bit	**Description**
31:29	**Reserved** Project: All Format:
28:16	**Horizontal_Blank_End** Project: All This field specifies Horizontal Blank End position relative to the horizontal active display start. The number of pixels within horizontal blank needs to be a multiple of two when driving the LVDS port in two channel mode. The minimum horizontal blank size is 32 pixels. This register must always be programmed to the same value as the Horizontal Total.
15:13	**Reserved** Project: All Format:
12:0	**Horizontal_Blank_Start** Project: All This field specifies the Horizontal Blank Start position relative to the horizontal active display start. This register must always be programmed to the same value as the Horizontal Active.

3.1.3 HSYNC—Horizontal Sync

HSYNC	
Register Type: MMIO **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 32 Write Protect by Panel Power Sequencer when panel is connected to this transcoder.	
Bit	**Description**
31:29	**Reserved** Project: All Format: MBZ
28:16	**Horizontal_Sync_End** Project: All Default Value: 0b This field specifies the Horizontal Sync End position relative to the horizontal active display start. It is programmed with HorizontalActive+FrontPorch+Sync-1 HDMI and DVI with audio are not supported when HSYNC Start is programmed equal to HBLANK Start. The number of pixels within horizontal sync needs to be a multiple of two when driving the LVDS port in two channel mode. This value must be greater than the horizontal sync start and less than Horizontal Total.
15:13	**Reserved** Project: All Format: MBZ
12:0	**Horizontal_Sync_Start** Project: All Default Value: 0b This field specifies the Horizontal Sync Start position relative to the horizontal active display start. It is programmed with HorizontalActive+FrontPorch-1 The number of pixels from active to horizontal sync needs to be a multiple of two when driving the LVDS port in two channel mode This value must be greater than Horizontal Active.

3.1.4 VTOTAL—Vertical Total

VTOTAL	
Register Type: MMIO **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 32 Write Protect by Panel Power Sequencer when panel is connected to this transcoder.	
Bit	**Description**
31:29	**Reserved** Project: All Format:
28:16	**Vertical_Total** Project: All This field specifies Vertical Total size. This should be equal to the sum of the vertical active and the vertical blank sizes. For progressive display modes, this field is programmed to the number of lines desired minus one. For interlaced display modes, this field is programmed with the number of lines desired minus two. The vertical counter is incremented on the leading edge of the horizontal sync. For interlaced display, hardware uses this value to calculate the vertical total in each field. Note that both even and off vertical totals are supported. This register must always be programmed to the same value as the Vertical Blank End.
15:12	**Reserved** Project: All Format:
11:0	**Vertical_Active** Project: All This field specifies Vertical Active Display size. Note that the first vertical active display line is considered pixel number 0. This field is programmed to the number of lines desired minus one. When using the internal panel fitting logic, the minimum vertical active area must be seven lines. For interlaced display, hardware uses this value to calculate the vertical active in each field. This register must always be programmed to the same value as the Vertical Blank Start.

3.1.5 VBLANK—Vertical Blank

VBLANK	
Register Type: MMIO **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 32 Write Protect by Panel Power Sequencer when panel is connected to this transcoder.	
Bit	**Description**
31:29	**Reserved** Project: All Format:
28:16	**Vertical_Blank_End** Project: All This field specifies Vertical Blank End position relative to the vertical active display start. The minimum vertical blank size is 5 lines. For interlaced display, hardware uses this value to calculate the vertical blank end in each field. This register must always be programmed to the same value as the Vertical Total.
15:13	**Reserved** Project: All Format:
12:0	**Vertical_Blank_Start** Project: All This field specifies the Vertical Blank Start position relative to the vertical active display start. For interlaced display, hardware uses this value to calculate the vertical blank start in each field. This register must always be programmed to the same value as the Vertical Active

3.1.6 VSYNC—Vertical Sync

VSYNC	
Register Type: MMIO **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 32 Write Protect by Panel Power Sequencer when panel is connected to this transcoder.	
Bit	**Description**
31:29	**Reserved** Project: All Format:
28:16	**Vertical_Sync_End** Project: All This field specifies the Vertical Sync End position relative to the vertical active display start. It is programmed with VerticalActive+FrontPorch+Sync-1 For interlaced display, hardware uses this value to calculate the vertical sync start in each field. This value must be greater than the vertical sync start and less than Vertical Total.
15:13	**Reserved** Project: All Format:
12:0	**Vertical_Sync_Start** Project: All This field specifies the Vertical Sync Start position relative to the vertical active display start. It is programmed with VerticalActive+FrontPorch-1 For interlaced display, hardware uses this value to calculate the vertical sync end in each field. This value must be greater than Vertical Active.

3.1.7 VSYNCSHIFT— Vertical Sync Shift

VSYNCSHIFT	
Register Type: MMIO **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 32 Write Protect by Panel Power Sequencer when panel is connected to this transcoder.	
Bit	**Description**
31:13	**Reserved** Project: All Format:
12:0	**Second_Field_VSync_Shift** Project: All This value specifies the vertical sync alignment for the start of the interlaced second field, expressed in terms of the absolute pixel number relative to the horizontal active display start. This value will only be used if the transcoder is in an interlaced mode. Typically, the interlaced second field vertical sync should start one pixel after the point halfway between successive horizontal syncs, so the value of this register should be programmed to: horizontal sync start - floor[horizontal total / 2] (use the actual horizontal sync start and horizontal total values and not the minus one values programmed into the registers) Legacy operation can be simulated by programming a value of floor[htotal / 2] (use the programmed minus one horizontal total value). This vertical sync shift only occurs during the interlaced second field. In all other cases the vertical sync start position is aligned with horizontal sync start.

3.2 Transcoder M/N Values

These values are used for DisplayPort.

When switching between two refresh rates, both the M1/N1 data and link values and the M2/N2 link values must be programmed. For dynamic refresh rate control, M1/N1 values are the primary values and are used for the normal M/N value setting, and M2/N2 values are the secondary values and are used for the lower power M/N value setting. Selection of M1/N1 or M2/N2 is indicated via MSA from the CPU display.

Example calculation of TU, Data M, and Data N: (See DisplayPort specification for exact calculation)

For modes that divide into the link frequency evenly,

Active/TU = Payload/Capacity = Data M/N = dot clock * bytes per pixel / ls_clk * number of lanes

Default value to program TU size is "111111" for TU size of 64.

Calculation of Link M and Link N:

Link M/N = dot clock / ls_clk

Restriction on clocks and number of lanes:
Number of lanes >= INT(dot clock * bytes per pixel / ls_clk)
Pcdclk * number of lanes >= dot clock * bytes per pixel

Please note that in the DisplayPort specification, dot clock is referred to as strm_clk.

3.2.1 DATAM— Data M Value

DATAM	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank
Double Buffer Armed By:	Writing the LINKN

Bit	Description
31	**Reserved** Project: All Format: MBZ
30:25	**TU_Size** Project: All This field is the size of the transfer unit, minus one.
24	**Reserved** Project: All Format: MBZ
23:0	**Data_M_value** Project: All This field is the m value for internal use of the DDA.

3.2.2 DATAN— Data N Value

DATAN	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank
Double Buffer Armed By:	Writing the LINKN

Bit	Description
31:24	**Reserved** Project: All Format: MBZ
23:0	**Data_N_value** Project: All This field is the n value for internal use of the DDA.

3.2.3 LINKM— Link M Value

LINKM	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank
Double Buffer Armed By:	Writing the LINKN

Bit	Description
31:24	**Reserved** Project: All Format: MBZ
23:0	**Link_M_value** Project: All This field is the m value for external transmission in the Main Stream Attributes.

3.2.4 LINKN— Link N Value

LINKN	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank

Writes to this register arm M/N registers for this transcoder.

Bit	Description
31:24	**Reserved** Project: All Format: MBZ
23:0	**Link_N_value** Project: All This field is the n value for external transmission in the Main Stream Attributes and VB-ID.

3.3 Transcoder Video DIP

3.3.1 VIDEO_DIP_CTL—Video DIP Control

VIDEO_DIP_CTL	
Register Type:	MMIO
Project:	All
Default Value:	20000900h
Access:	R/W
Size (in bits):	32

Please note that writes to this register take effect immediately. Therefore, it is critical for software to follow the write and read sequences as described in the bit 31 text.

Bit	Description
31	**Enable_Graphics_DIP** Project: All Default Value: 0b Data Island Packet (DIP) is a mechanism that allows up to 36 bytes to be sent over digital port during VBLANK, according to the HDMI and DisplayPort specifications. This includes header, payload, checksum and ECC information. Each type of DIP can be sent once per vsync, once every other vsync, or once. This data can be transmitted on either transcoder, through any digital port (digital port B, C or D), but not two simultaneously on one transcoder. Please note that the audio subsystem is also capable of sending Data Island Packets. These packets are programmed by the audio driver and can be read by in MMIO space via the audio control state and audio HDMI widget data island registers. Write sequence: 1. Wait for 1 VSync to ensure completion of any pending DIP transmissions. 2. Disable the Video_DIP_type_enable and set the Video_DIP_buffer_index for the DIP being written. 3. Set the Video_DIP_access_address to the desired DWORD to be written. 4. Write DIP data 1 DWORD at a time. The DIP access address auto-increments with each DWORD write, wrapping around to address 0 when the max buffer address size has been reached. Please note that software must write an entire DWORD at a time. 5. Enable the DIP type and transmission frequency. Read sequence: 1. Set the Video_DIP_buffer_index for the DIP being read. 2. Set the Video_DIP_access_address to the desired DWORD to be read. 3. Read DIP data 1 DWORD at a time. The DIP access address auto-increments with each DWORD read, wrapping around to address 0 when the max buffer address size has been reached.

VIDEO_DIP_CTL

Register Type: MMIO
Project: All
Default Value: 20000900h
Access: R/W
Size (in bits): 32

Please note that writes to this register take effect immediately. Therefore, it is critical for software to follow the write and read sequences as described in the bit 31 text.

Bit	Description

Value	Name	Description	Project
0b	Disable	Video DIP is disabled	All
1b	Enable	Video DIP is enabled	All

Programming Notes

- Partial DIPs are never sent out while the port is enabled. Disabling the DIP at the same time it is being transferred will result in the DIP being completed before the function is disabled.
- Shutting off the port on which DIP is being transmitted will result in partial transfer of DIP data. There is no need to switch off the DIP enable bit if the port transmitting DIP is disabled.
- When disabling both the DIP port and DIP transmission, first disable the port and then disable DIP.
- Enabling a DIP function at the same time that the DIP would have been sent out (had it already been enabled) will result in the DIP being sent on the following frame.
- Enabling should only be done after the buffer contents have been written.
- If DIP is enabled but DIP types are all disabled, no DIP is sent. However, a single Null DIP will be sent at the same point in the stream that DIP packets would have been sent. This is done to keep the port in HDMI mode, otherwise it would revert to DVI mode. HDMI_CTL Null_packets_enabled_during_vsync overrides this behavior.
- Enable_Graphics_DIP (bit 31) and Data_Island_Packet_type_enable for AVI (bit 23) must be set or cleared in the same write if the HDMI port is already enabled

30:26 **Reserved** Project: All Format:

25 **GCP_DIP_enable**

Project: All

Default Value: 0b

This bit enables the output of the General Control Packet. GCP is different from other DIPs in that much of the payload is automatically reflected in the packet, and therefore a DIP buffer for GCP is not needed. Please refer to the GCP payload register for payload details. Writes to this bit take effect immediately.

Software must enable this bit before enabling the port when GCP is required, and disable this bit after disabling the port.

Value	Name	Description	Project
0b	Disable	GCP DIP disabled	All
1b	Enable	GCP DIP enabled	All

VIDEO_DIP_CTL

Register Type: MMIO
Project: All
Default Value: 20000900h
Access: R/W
Size (in bits): 32

Please note that writes to this register take effect immediately. Therefore, it is critical for software to follow the write and read sequences as described in the bit 31 text.

Bit	Description
24:21	**Video_DIP_type_enable** Project: All Default Value: 0001b Enable AVI DIP These bits enable the output of a given data island packet type. It can be updated while the port is enabled and is immediately updated (not double-buffered). Within 2 vblank periods, the DIP is guaranteed to have been transmitted. Software must enable VS DIP type (initialized to zero or programmed with valid VS payload) in addition when it is desired to enable AVI+SPD+GMP AVI enable (bit 23) and Enable_Graphics_DIP (bit 31) must be set or cleared in the same write if the HDMI port is already enabled. AVU should be updated dynamically (without clearing the enable bit) by waiting for vertical blank and then updating the AVI buffer.

Value	Name	Description	Project
XXX1b	Enable AVI	Enable AVI DIP	All
XX1Xb	Enable Vendor	Enable Vendor-specific DIP	All
X1XXb	Enable Gamut	Enable Gamut Metadata Packet	All
1XXXb	Enable Source	Enable Source Product Description DIP	All

Bit	Description
20:19	**Video_DIP_buffer_index** Project: All Default Value: 00b This field is used during programming of different DIPs. These bits are used as an index to their respective DIP buffers. The transmission frequency must also be written when programming the buffer.

Value	Name	Description	Project
00b	AVI	AVI DIP (31 bytes of space available)	All
01b	Vendor-specific	Vendor-specific DIP	All
10b	Gamut Metadata	Gamut Metadata Packet	All
11b	Source Product	Source Product Description DIP	All

Bit	Description
18	**Reserved** Project: All Format:

VIDEO_DIP_CTL

Register Type: MMIO
Project: All
Default Value: 20000900h
Access: R/W
Size (in bits): 32

Please note that writes to this register take effect immediately. Therefore, it is critical for software to follow the write and read sequences as described in the bit 31 text.

Bit	Description
17:16	**Video_DIP_frequency** Project: All Default Value: 00b These bits dictate the frequency of Video DIP transmission for the DIP buffer index designated in bits 20:19. When writing Video DIP data, this value is also latched when the first DW of the Video DIP is written. When read, this value reflects the Video DIP transmission frequency for the Video DIP buffer designated in bits 20:19. This field shall be ignored for Gamut Metadata Packet transmission. Always program AVI to "Send Every Vsync" when enabling AVI.

Value	Name	Description	Project
00b	Send Once	Send Once	All
01b	Every VSync	Send Every VSync (Default for AVI)	All
10b	Every Other Vsync	Send at least every other VSync	All
11b	Reserved	Reserved	All

Bit	Description
15:12	**Reserved** Project: All Format: MBZ
11:8	**Video_DIP_buffer_size** Project: All Access: Read Only Default Value: 1001b This reflects the buffer size in dwords available for the type of Video DIP being indexed by bits 20:19 of this register, including the header. It is hardwired to the maximum size of a Video DIP, 36 bytes. Please note that this count includes ECC bytes, which are not writable by software. These bits are immediately valid after write of the DIP index.
7:4	**Reserved** Project: All Format: MBZ
3:0	**Video_DIP_access_address** Project: All Selects the DWORD address for access to the Video DIP buffers. This value is automatically incremented after each read or write of the Video DIP Data Register. The value wraps back to zero when it autoincrements past the max address value of 0xF. This field change takes effect immediately after being written. The read value indicates the current access address.

3.3.2 VIDEO_DIP_DATA–Video Data Island Packet Data

VIDEO_DIP_DATA	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access:	R/W (DWORD access only, no byte access)
Size (in bits):	32
Bit	**Description**
31:0	**Video_DIP_DATA** Project: All When read, this returns the current value at the location specified in the Video DIP buffer index select and Video DIP access address fields. The address index is incremented after each read or write of this register. DIP data can be read at any time. Data should be loaded before enabling the transmission through the DIP type enable bit.

Construction of DIP Data:

Dword	Byte3	Byte2	Byte1	Byte0
0	DP : HB3 HDMI: ECC (RO)	HB2	HB1	HB0
1	DB3	DB2	DB1	DB0
2	DB7	DB6	DB5	DB4
3	DB11	DB10	DB9	DB8
4	DB15	DB14	DB13	D12
5	DB19	DB18	DB17	DB16
6	DB23	DB22	DB21	DB20
7	DB27	DB26	DB25	DB24
8	ECC byte3 (RO)	ECC byte2 (RO)	ECC byte1 (RO)	ECC byte0 (RO)
9	DP: ECC byte7 (RO) HDMI : Reserved	DP: ECC byte6 (RO) HDMI : Reserved	DP: ECC byte5 (RO) HDMI : Reserved	DP: ECC byte4 (RO) HDMI : Reserved
10	DP: ECC byte11 (RO) HDMI : Reserved	DP: ECC byte10 (RO) HDMI : Reserved	DP: ECC byte9 (RO) HDMI : Reserved	DP: ECC byte8 (RO) HDMI : Reserved
11	Reserved	Reserved	Reserved	Reserved
12	Reserved	Reserved	Reserved	Reserved

Dword	Byte3	Byte2	Byte1	Byte0
13	Reserved	Reserved	Reserved	Reserved
14	Reserved	Reserved	Reserved	Reserved
15	Reserved	Reserved	Reserved	Reserved

HB = Header Byte

DB = Data Byte

DP = DisplayPort

3.3.3 VIDEO_DIP_GCP–Video Data Island Payload

VIDEO_DIP_GCP

Register Type: MMIO
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 32

Bit	Description
31:3	**Reserved** Project: All Format: MBZ
2	**GCP_color_indication** Project: All Default Value: 0b This bit must be set when in deep color mode. It may optionally be set for 24-bit mode. It must be set if the sink attached to the transcoder can receive GCP data.

Value	Name	Description	Project
0b	Don't Indicate	Don't indicate color depth. CD and PP bits in GCP set to zero	All
1b	Indicate	Indicate color depth using CD bits in GCP. It will be set depending on programmed pixel depth in port control register	All

VIDEO_DIP_GCP

1

GCP_default_phase_enable

Project: All

Default Value: 0b

Indicates the video timings meet alignment requirements such that the following conditions are met:

1) Htotal is an even number

2) Hactive is an even number

3) Hsync is an even number

4) Front and back porches for Hsync are even numbers

5) Vsync always starts on an even-numbered pixel within a line in interlaced modes (starting counting with 0)

Value	Name	Description	Project
0b	Clear	Default phase bit in GCP is cleared	All
1b	Require Met	Default phase bit in GCP is set. All requirements must be met before setting this bit	All

0

GCP_AV_mute

Project: All

Default Value: 0b

Set AV mute bit in GCP.

Value	Name	Description	Project
0b	Clear	AV mute bit in GCP is cleared. When this bit transitions to 0, the AV mute clear flag is sent in the next GCP packet	All
1b	Set	AV mute bit in GCP is set. When this bit transitions to 1, the AV mute set flag is sent in the next GCP packet	All

3.4 Transcoder DisplayPort Control

3.4.1 TRANS_DP_CTL—Transcoder DisplayPort Control

TRANS_DP_CTL	
Register Type:	MMIO
Project:	All
Default Value:	60000018h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Depends on bit

This register configures transcoder based DisplayPort logic. This register must be used in conjunction with the DisplayPort Control registers.

Bit	Description
31	**Transcoder_DP_Output_Enable** Project: All Default Value: 0b This bit determines if this transcoder will output to a DisplayPort. Transcoder_DP_Port_Select needs to be set to 11b (None) when writing a 0b to this bit to disable transcoder output to DisplayPort.

Value	Name	Description	Project
0b	Disable	Disable the transcoder output to DisplayPort	All
1b	Enable	Enable the transcoder to output to DisplayPort	All

Bit	Description
30:29	**Transcoder_DP_Port_Select** Project: All Default Value: 11b Reserved This bit determines which DisplayPort port block will be driven by the DisplayPort output of this transcoder. Port selection takes place on the Vblank after being written. This field needs to be set to 11b (None) when writing a 0b to Transcoder_DP_Output_Enable to disable transcoder output to DisplayPort.

Value	Name	Description	Project
00b	Port B	DisplayPort Port B	All
01b	Port C	DisplayPort Port C	All
10b	Port D	DisplayPort Port D	All
11b	None	No port selected	All

Bit	Description				
28:27	**Reserved**	Project:	All	Format:	MBZ
26:19	**Reserved**	Project:	All	Format:	MBZ

TRANS_DP_CTL

18 — **Transcoder_DP_Enh_Framing**

Project: All

Default Value: 0b

This bit selects enhanced framing on the DisplayPort ouput of this transcoder.

Locked once port is enabled. Updates when the port is disabled then re-enabled

Value	Name	Description	Project
0b	Disable	Enhanced framing disabled	All
1b	Enable	Enhanced framing enabled	All

17:12 — **Reserved** Project: All Format: MBZ

11:9 — **Transcoder_DP_Bits_Per_Color**

Project: All

Default Value: 000b

This field selects the number of bits per color output on DisplayPorts connected to this transcoder. Software should enable dithering in the pipe if selecting a pixel color depth higher or lower than the pixel color depth of the frame buffer.

Value	Name	Description	Project
000b	8 bpc	8 bits per color	All
001b	10 bpc	10 bits per color	All
010b	6 bpc	6 bits per color	All
011b	12 bpc	12 bits per color	All
Others	Reserved	Reserved	All

8:5 — **Reserved** Project: All Format: MBZ

4:3 — **Transcoder_DP_Sync_Polarity**

Project: All

Default Value: 11b VS and HS are active high

Indicates the polarity of Hsync and Vsync to be transmitted in MSA on this transcoder DisplayPort output.

Value	Name	Description	Project
00b	Low	VS and HS are active low (inverted)	All
01b	VS Low, HS High	VS is active low (inverted), HS is active high	All
10b	VS High, HS Low	VS is active high, HS is active low (inverted)	All
11b	High	VS and HS are active high	All

2:0 — **Reserved** Project: All Format: MBZ

3.5 Analog Port CRT DAC

3.5.1 DAC_CTL—Analog Port CRT DAC Control

DAC_CTL

Register Type: MMIO
Project: All
Default Value: 00040000h
Access: R/W
Size (in bits): 32

Bit	Description
31	**Port_Enable** Project: All Default Value: 0b This bit enables or disables the analog port CRT DAC and syncs outputs.

Value	Name	Description	Project
0b	Disable	Disable the analog port DAC and disable output of syncs	All
1b	Enable	Enable the analog port DAC and enable output of syncs	All

Bit	Description
30:29	**Transcoder_Select** Project: All Default Value: 00b Determines which transcoder will feed this DAC port.

Value	Name	Description	Project
00b	Transcoder A	Transcoder A	All
01b	Transcoder B	Transcoder B	All
10b	Transcoder C	Transcoder C	All
11b	Reserved	Reserved	All

Bit	Description
28:26	**Reserved** Project: All Format:

DAC_CTL

25:24

CRT_HPD_Channel_Status

Project: All

Access: Read Only

Default Value: 00b

These bits are set when a CRT hot plug or unplug event has been detected and indicate which color channels were attached. Write a one to these bits to clear the status. The rising or falling edges of these bits are ORed together to go to the SDE_ISR CRT hot plug register bit.

Value	Name	Description	Project
00b	None	No channels attached	All
01b	Blue	Blue channel only is attached	All
10b	Green	Green channel only is attached	All
11b	Both	Both blue and green channel attached	All

23

CRT_HPD_Enable

Project: All

Default Value: 0b

Hot plug detection is used to set status bits or an interrupt on the connection or disconnection of a CRT to the analog port CRT DAC.

Value	Name	Description	Project
0b	Disable	CRT hot plug detection is disabled	All
1b	Enable	CRT hot plug detection is enabled	All

22

CRT_HPD_Activation_Period

Project: All

Default Value: 0b

This bit sets the activation period for the CRT hot plug circuit. [DevCPT] rawclk=125mhz

Value	Name	Description	Project
0b	64 rawclk	64 rawclk periods	All
1b	128 rawclk	128 rawclk periods	All

21

CRT_HPD_Warmup_Time

Project: All

Default Value: 0b

This bit sets the warmup time for the CRT hot plug circuit. [DevCPT] rawclk=125mhz

Value	Name	Description	Project
0b	4ms	Approximately 4ms	All
1b	8ms	Approximately 8ms	All

DAC_CTL

20

CRT_HPD_Sampling_Period

Project: All

Default Value: 0b

This bit determines the length of time between sampling periods when the transcoder is disabled.

Value	Name	Description	Project
0b	2 seconds	Approximately 2 seconds	All
1b	4 seconds	Approximately 4 seconds	All

19:18

CRT_HPD_Voltage_Value

Project: All

Default Value: 01b 50

Compare value for Vref to determine whether the analog port is connected to a CRT.

Value	Name	Description	Project
00b	40	40	All
01b	50	50 (Default)	All
10b	60	60	All
11b	70	70 (bit 17 must be = 1)	All

17

CRT_HPD_Reference_Voltage

Project: All

Default Value: 0b

Value	Name	Description	Project
0b	325mv	325mv	All
1b	475mv	475mv (bits 19:18 must be = 11)	All

16

Force_CRT_HPD_Trigger

Project: All

Default Value: 0b

Triggers a CRT hotplug/unplug detection cycle independent of the hot plug detection enable bit. This bit is automatically cleared after the detection is completed. The result of this trigger is reflected in the CRT Hot Plug Detection Status. Software must reset status after a force CRT detect trigger.

Value	Name	Description	Project
0b	No Trigger	No Trigger	All
1b	Force Trigger	Force Trigger	All

15:5

Reserved Project: All Format:

DAC_CTL	
4	**VSYNC_Polarity_Control** Project: All Default Value: 0b The output VSYNC polarity is controlled by this bit. This is used to implement display modes that require inverted polarity syncs and to set the disabled state of the VSYNC signal.

Value	Name	Description	Project
0b	Low	Active Low	All
1b	High	Active High	All

3	**HSYNC_Polarity_Control** Project: All Default Value: 0b The output HSYNC polarity is controlled by this bit. This is used to implement display modes that require inverted polarity syncs and to set the disabled state of the HSYNC signal.

Value	Name	Description	Project
0b	Low	Active Low	All
1b	High	Active High	All

2:0	**Reserved** Project: All Format:

3.6 HDMI Port

3.6.1 HDMI_CTL—HDMI Port Control

HDMI_CTL	
Register Type:	MMIO
Project:	All
Default Value:	00000018h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Depends on bit

The operating mode (HDMI/DVI) of this port is determined by the setting of the encoding register field. **Ports C and D can only be used in HDMI/DVI mode.**

Note that DisplayPort B uses the same physical pins as HDMI/DVI B. Therefore HDMI/DVI B and DisplayPort B cannot be enabled simultaneously. The same applies for ports C and D.

Bit	Description
31	**Port_Enable** Project: All

HDMI_CTL

Default Value: 0b

Disabling this port will put it in its lowest power state. Port enable takes place on the Vblank after being written. Both this bit and the audio output bit of this register must be enabled to send audio over this port.

Value	Name	Description	Project
0b	Disable	Disable and tristates the port interface	All
1b	Enable	Enables the port interface	All

30:29

Transcoder_Select

Project: All

Default Value: 00b

This bit determines from which display transcoder the source data will originate. Transcoder selection takes place on the Vblank after being written.

Value	Name	Description	Project
00b	Transcoder A	Transcoder A	All
01b	Transcoder B	Transcoder B	All
10b	Transcoder C	Transcoder C	All
11b	Reserved	Reserved	All

28:26

Color_Format

Project: All

Default Value: 000b

This field selects the number of bits per color sent to a receiver device connected to this port. Color format takes place on the Vblank after being written. Color format change must be done as a part of mode set since different color depths require different pixel clock settings.

Software should enable dithering in the pipe/transcoder if selecting a pixel color depth higher or lower than the pixel color depth of the frame buffer.

Value	Name	Description	Project
000b	8 bpc	8 bits per color	All
011b	12 bpc	12 bits per color	All
Others	Reserved	Reserved	All

25

Reserved

Project: All

24

Reserved Project: All Format:

23

Reserved

Project: All

22:19

Reserved Project: All Format:

18

Reserved

Project: All

HDMI_CTL

17:16	**Reserved** Project: All Format:
15	**Port_Lane_Reversal** Project: All Default Value: 0b This bit reverses the order of the 4 lanes within the port. It is an OEM configurable feature. Locked once port is enabled. Updates when the port is disabled then re-enabled

Value	Name	Description	Project
0b	Not reversed	Not reversed	All
1b	Reversed	Reversed	All

14:12	**Reserved** Project: All Format:
11:10	**Encoding** Project: All Format: These bits select among encoding types. It is set as part of the display detection process. Control codes for ANSI 8b/10b and TMDS encoding must be programmed using these bits.

Value	Name	Description	Project
00b	Reserved		All
10b	TMDS	TMDS encoding for HDMI/DVI. See the HDMI specification for control codes. In this mode, the external HPD pin is used to generate hotplug.	All
Others	Reserved	Reserved	All

9	**HDMI_or_DVI_Select** Project: All Default Value: 0b This bit selects between HDMI and DVI modes of operation. HDMI mode enables a null packet (32 bytes of a value of 0) to be sent when Vsync=1 on this port, required for HDMI operation. It also enables preambles and guardbands prior to the null packets, in accordance with the HDMI specification. It is only valid for modes that use TMDS encoding.

Value	Name	Description	Project
0b	DVI	Port will function in DVI mode if no DIP packets are enabled and no audio is present.	All
1b	HDMI	Port will function in HDMI mode.	All

8	**Reserved** Project: All Format: MBZ
7	**Reserved** Project: All

HDMI_CTL

6	**Audio_Output_Enable** Project: All Default Value: 0b This bit directs audio to this port. When enabled and audio data is available, the audio data will be combined with the video data and sent over this port. The audio unit uses the status of this bit to indicate presence of the HDMI output to the audio driver.

Value	Name	Description	Project
0b	Disable	No audio output on this port	All
1b	Enable	Enable audio on this port	All

5	**Reserved** Project: All
4:3	**Sync_Polarity** Project: All Default Value: 11b VS and HS are active high Indicates the polarity of Hsync and Vsync. Inverted polarity is transmitted as SYNC-BLANK-SYNC and standard polarity is transmitted as BLANK-SYNC-BLANK. For example, if Vsync is not inverted and Hsync is inverted, an Hsync period transmitted during Vsync would be transmitted as BLANK+VS+HS – BLANK+VS – BLANK+VS+HS.

Value	Name	Description	Project
00b	VS Low, HS Low	VS and HS are active low (inverted)	All
01b	VS Low, HS High	VS is active low (inverted), HS is active high	All
10b	VS High, HS Low	VS is active high, HS is active low (inverted)	All
11b	VS High, HS High	VS and HS are active high (Default)	All

2	**Port_Detected** Project: All Access: Read Only Default Value: 0b Read-only bit indicating whether a digital display was detected during initialization. It signifies the level of the GMBUS port (port 4 for B, port 3 for C, port 5 for D) data line at boot. This bit is valid regardless of whether the port is enabled.

Value	Name	Description	Project
0b	Not Detected	Port not detected during initialization	All
1b	Detected	Port detected during initialization	All

1:0	**Reserved** Project: All Format: MBZ

3.7 LVDS

3.7.1 LVDS—LVDS Port Control Register

LVDS—LVDS Port Control Register

Register Type: MMIO
Project: All
Default Value: 00000000h
Access: R/W Protect
Size (in bits): 32
Register Type: MMIO

Write Protect by Panel Power Sequencer

Bit	Description
31	**LVDS_Port_Enable** Project: All Default Value: 0b When disabled the LVDS port is inactive and in it's low power state. Enabling the LVDS port changes the way that the PLL for this transcoder is programmed. This bit must be set before the display PLL is enabled and the port is power sequenced on using the panel power sequencing logic. LVDS clockgating must be disabled (South Display Unit Clock Gating Disable bits #30, 14 in 0xC2020 set to '1') at boot and remained disabled before enabling dual channel LDVS mode.

Value	Name	Description	Project
0b	Disable	The port is disabled and all LVDS pairs are powered down.	All
1b	Enable	The port is enabled (port must be enabled before powering up a connected panel)	All

Bit	Description
30:29	**LVDS_Port_Transcoder_Select** Project: All Default Value: 1b Transcoder B

Value	Name	Description	Project
0b	Transcoder A	The port gets data from Transcoder A	All
1b	Transcoder B	The port gets data from Transcoder B	All

Bit	Description
28:25	**Reserved** Project: All Format:

LVDS—LVDS Port Control Register

24

Data_Format_Select

Project: All

Default Value: 0b

Combined with the other control bits it selects the LVDS data format. Other control bits in this register determine if two channel is enabled and 18 or 24 bit color is enabled.

Value	Name	Description	Project
0b	1x18.0, 2x18.0, 1x24.0 or 2x24.0	1x18.0, 2x18.0, 1x24.0 or 2x24.0	All
1b	1x24.1 or 2x24.1	1x24.1 or 2x24.1	All

23

LE_Control_Enable

Project: All

Default Value: 0b

This bit is used when the second channel control signal field indicates that we are using the LE instead of HS and the two channel mode is enabled. In single channel mode, this bit has no effect.

Value	Name	Description	Project
0b	Send 0	Send 0 on second channel HS (B2<2>)	All
1b	Send 1	Send 1 on second channel HS	All

22

LF_Control_Enable

Project: All

Default Value: 0b

This bit is used when the second channel control signal field indicates that we are using the LF instead of VS and two channel mode is enabled. In single channel mode, this bit has no effect.

Value	Name	Description	Project
0b	Send 0	Send 0 on second channel VS (B2<3>)	All
1b	Send 1	Send 1 on second channel VS	All

21

VSYNC_Polarity

Project: All

Default Value: 0b

This controls the polarity of the VSYNC indicator that is sent over the LVDS connection. Panels may require one or the other polarity or work with either polarity.

Value	Name	Description	Project
0b	No Invert	No inversion (1=active)	All
1b	Invert	Invert the sense (0=active)	All

LVDS—LVDS Port Control Register

20	**HSYNC_Polarity(LP_Invert)** Project: All Default Value: 0b This controls the polarity of the HSYNC indicator that is sent over the LVDS connection. Panels may require one or the other polarity or work with either polarity.

Value	Name	Description	Project
0b	No Invert	No inversion (1=active)	All
1b	Invert	Invert the sense (0=active)	All

19	**DE_Invert** Project: All Default Value: 0b This controls the polarity of the DE indicator that is sent over the LVDS connection.

Value	Name	Description	Project
0b	No Invert	No inversion of DE (1=active)	All
1b	Invert	Invert the sense of DE (0=active)	All

18:17	**Second_Channel_Control_Signals** Project: All Default Value: 00b This bit only applies to the two channel modes of operation it has no effect in single channel modes.

Value	Name	Description	Project
00b		Send DE, HS, VS on second channel if enabled	All
01b	Reserved	Reserved	All
10b		Do not send DE, HS, VS on second channel use zero instead	All
11b		Use DE=0, HS=LE, VS=LF on second channel	All

16	**Channel_Reserved_Bits** Project: All Default Value: 0b

Value	Name	Description	Project
0b	Send 0	Send 0 for the channel reserved bits	All
1b	Send Duplicate	Send duplicate data bit for reserved bits	All

15:11	**Reserved** Project: All Format:

LVDS—LVDS Port Control Register

10

Buffer_Power_Down_State

Project: All

Default Value: 0b

This bit selects the state of the LVDS buffers during a powered down state caused by the power sequence logic power down. This selection will be made based on the connected panel requirements.

Value	Name	Description	Project
0b	Zero	Zero Volts (Driven on both lines of the pairs)	All
1b	Tri-State	Tri-State (High impedance state)	All

9:8

ClkA0_A2_Control

Project: All

Default Value: 00b

This field controls the A0-A2 data pairs and CLKA. It sets the highest level of activity that is allowed on these lines when the panel is powered on. Power sequencing for LVDS connected panels overrides the control. When the power sequencer is in the power down mode all signals are in the power down state.

Value	Name	Description	Project
00b	Power Down	Power Down all A channel signals including A3 (0V)	All
01b	Power Up Data 0	Power up – A0, A1, A2 Data bits forced to 0,Timing active, Clock Active	All
10b	Reserved	Reserved	All
11b	Power Up All Active	Power up – Data lines and clock active	All

7:6

Eight_bit_ch_A3_B3_Control

Project: All

Default Value: 00b

This field can control both the A3 and B3 data pairs. Enabling those pairs indicates the selection of 8-bit per color channel mode. It sets the highest level of activity that is allowed on these lines when the panel is powered on. The A3 pair will only be powered up if both this field and the A0, A1, A2, CLKA field indicates that the pair should be powered up and will only be active if both indicate that it should be active. The B3 pair will only be powered up if both this field and the B0, B1, B2, (B3) field indicates that the pair should be powered up and will only be active if both indicate that it should be active. Power sequencing for LVDS connected panels overrides the control. When the power sequencer is in the power down mode all signals are in the power down state.

Value	Name	Description	Project
00b	Power Down	Power Down all signals A3, B3 (common mode)	All
01b	Power Up Data 0	Power up – A3, (B3) Data (pixel data not control) lines forced to 0 output	All
10b	Reserved	Reserved	All
11b	Power Up Data Active	Power up – A3, (B3) Data lines active	All

LVDS—LVDS Port Control Register

5:4

Two_channel_mode_ClkB_Control

Project: All

Default Value: 00b

When in two channel mode, this field controls the CLKB pair. It sets the highest level of activity that is allowed on these lines when the panel is powered on. The CLKB pair should only be powered up if the B0, B1, B2, (B3) field indicates that the second channel should be powered up and will only be active if both indicate that it should be active. Power sequencing for LVDS connected panels overrides the control.

Value	Name	Description	Project
00b	Power Down	Power Down CLKB (common mode)	All
01b	Power Up CLKB 0	Power up – CLKB Forced to 0	All
10b	Reserved	Reserved	All
11b	Power Up CLKB Active	Power up – Clock B active	All

3:2

Two_channel_mode_B0_B2_Control

Project: All

Default Value: 00b

This field controls both the set B0-B2 data pairs. It sets the highest level of activity that is allowed on these lines when the panel is powered on. Power sequencing for LVDS connected panels overrides the control. During single channel operation (1x18.0), these bits need to be both zero. Two channel operation is selected by setting them to ones. Note that the second clock can be optionally enabled or disabled by the two channel mode ClkB control field.

Value	Name	Description	Project
00b	Power Down	Power Down all signals including B3 and CLKB	All
01b	Power Up Data 0	Power up – B0, B1, B2, Data lines forced to 0, timing is active	All
10b	Reserved	Reserved	All
11b	Power Up Data Active	Power up – Data lines active (color and timing)	All

1

Port_Detected

Project: All

Access: Read Only

Default Value: 0b

Read-only bit indicating whether LVDS was detected during initialization. It signifies the level of the GMBUS port 2 (LVDS) data line at boot. This bit is valid regardless of whether the port is enabled.

Value	Name	Description	Project
0b	Not Detected	LVDS not detected during initialization	All
1b	Detected	LVDS detected during initialization	All

0 **Reserved** Project: All Format:

3.8 DisplayPort

3.8.1 DP_CTL—DisplayPort Control

DP_CTL

Register Type:	MMIO
Project:	All
Default Value:	00000018h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Depends on bit

Port enable is write protected by Panel Power Sequencer when panel is connected to this port.

Note that DisplayPort B uses the same physical pins as HDMI/DVI B. Therefore HDMI/DVI B and DisplayPort B cannot be enabled simultaneously. The same applies for ports C and D.

Bit	Description
31	**DisplayPort_Enable** Project: All Default Value: 0b Disabling this port will put it in its lowest power state. Port enable takes place on the Vblank after being written. **[DevCPT-A, DevCPT-B0]** When panel power sequencing is selected for DisplayPort D, the clock gating disable register SCLKGATE_DIS 0xC2020 bit 14 must be set to 1b prior to enabling DisplayPort D, then cleared to 0b after disabling DisplayPort D.
30:28	**Reserved** Project: All Format: MBZ
27:25	**Voltage_swing_level_set** Project: All Default Value: 0b These bits are used for setting the voltage swing for pattern 1, defined as Vdiff_pp in the DisplayPort specification.

Value	Name	Description	Project
0b	Disable	Disable and tristate the DisplayPort interface	All
1b	Enable	Enable the DisplayPort interface	All

Value	Name	Description	Project
000b	0.4V	0.4V	All
001b	0.6V	0.6V	All
010b	0.8V	0.8V	All
011b	1.2V	1.2V	All
Others	Reserved	Reserved	All

DP_CTL

24:22 Preemphasis_level_set

Project: All

Default Value: 0b

These bits are used for setting link pre-emphasis for pattern 2, as defined in the DisplayPort specification.

Value	Name	Description	Project
000b	0dB	No pre-emphasis	All
001b	3.5dB	3.5dB pre-emphasis (1.5x)	All
010b	6dB	6dB pre-emphasis (2x)	All
011b	9.5dB	9.5dB pre-emphasis (3x)	All
Others	Reserved	Reserved	All

21:19 Port_Width_Selection

Project: All

Default Value: 0b

This bit selects the number of lanes to be enabled on the DisplayPort link. Port width change must be done as a part of mode set. **Locked once port is enabled. Updates when the port is disabled then re-enabled.**

Value	Name	Description	Project
000b	x1	x1 Mode	All
001b	x2	x2 Mode	All
011b	x4	x4 Mode	All
Others	Reserved	Reserved	All

18:16 Reserved Project: All Format: MBZ

15 Port_reversal

Project: All

Default Value: 0b

Enables lane reversal within the port: lane 0 mapped to lane 3, lane 1 mapped to lane 2, etc. Port reversal does not affect AUX channel lane mapping.

Locked once port is enabled. Updates when the port is disabled then re-enabled

Value	Name	Description	Project
0b	Not Reversed	Port not reversed	All
1b	Reversed	Port reversed	All

14:11 Reserved Project: All Format: MBZ

DP_CTL

10:8 **Link_training_pattern_enable**

Project: All

Default Value: 0b

These bits are used for link initialization as defined in the DisplayPort specification. Please note that the link must first be configured prior to sending training patterns.

When enabling the port, it must be turned on with pattern 1 enabled. When retraining, the port must be disabled, then re-enabled with pattern 1 enabled.

Value	Name	Description	Project
000b	Pattern 1	Pattern 1 enabled: Repetition of D10.2 characters	All
001b	Pattern 2	Pattern 2 enabled: Repetition of K28.5, D11.6, K28.5, D11.6, D10.2, D10.2, D10.2, D10.2, D10.2, D10.2. Please note that the entire pattern must complete before another pattern is sent. Scrambling initialization and disparity init commence at the end of the last iteration of pattern 2.	All
010b	Idle	Idle Pattern enabled: Transmit BS followed by VB-ID with NoVideoStream_flag set to 1, five times	All
011b	Normal	Link not in training: Send normal pixels	All
Others	Reserved	Reserved	All

7 **Reserved**

Project: All

6 **Audio_Output_Enable**

Project: All

Default Value: 0b

This bit enables audio output on this port. It may be enabled or disabled only when the link training is complete and set to "Normal".

Value	Name	Description	Project
0b	Disable	Audio output disabled	All
1b	Enable	Audio output enabled	All

5 **Reserved**

Project: All

4:3 **Reserved** Project: All Format: MBZ

DP_CTL	
2	**Port_Detected** Project: All Access: Read Only Default Value: 0b Read-only bit indicating whether a digital display was detected during initialization. It signifies the level of the detect pin (GMBUS port 4 for port B, GMBUS port 3 for port C, GMBUS port 5 for port D) at boot. This bit is valid regardless of whether the port is enabled. **Value** | **Name** | **Description** | **Project** 0b | Not Detected | Digital display not detected during initialization | All 1b | Detected | Digital display detected during initialization | All
1:0	**Reserved** Project: All Format: MBZ

3.8.2 DP_AUX_CTL—DisplayPort AUX Channel Control

DP_AUX_CTL	
Register Type:	MMIO
Project:	All
Default Value:	00050000h
Access:	R/W Special
Size (in bits):	32
Bit	**Description**
31	**Send_Busy** Project: All Default Value: 0b Setting this bit to a one initiates the transaction, when read this bit will be a 1 until the transmission completes. The transaction is completed when the response is received or when a timeout occurs. Do not write a 1 again until transaction completes. Writes of 0 will be ignored. **Programming Notes** Do not change any fields while Send/Busy bit 31 is asserted.
30	**Done** Project: All Access: R/W Clear A sticky bit that indicates the transaction has completed. SW must write a 1 to this bit to clear the event.
29	**Interrupt_on_Done** Project: All Format: Enable an interrupt in the hotplug status register when the transaction completes or times out.
28	**Time_out_error** Project: All Access: R/W Clear A sticky bit that indicates the transaction has timed out. SW must write a 1 to this bit to clear the event.

DP_AUX_CTL

27:26 — **Time_out_timer_value**

Project: All

Default Value: 00b

Used to determine how long to wait for receiver response before timing out.

Value	Name	Description	Project
00b	400us	400us	All
01b	600us	600us	All
10b	800us	800us	All
11b	1600us	1600us	All

25 — **Receive_error** Project: All Access: R/W Clear

A sticky bit that indicates that the data received was corrupted, not in multiples of a full byte, or more than 20 bytes. SW must write a 1 to this bit to clear the event.

24:20 — **Message_Size** Project: All Format:

This field is used to indicate the total number bytes to transmit (including the header). It also indicates the number of bytes received in a transaction (including the header). This field is valid only when the done bit is set, and if timeout or receive error has not occurred. Sync/Stop are not part of the message or the message size.

Reads of this field will give the response message size.

The read value will not be valid while Send/Busy bit 31 is asserted.

Message sizes of 0 or >20 are not allowed.

19:16 — **Precharge_Time** Project: All Format:

Default Value: 0101b 10us

Used to determine the precharge time for the Aux Channel drivers. During this time the Aux Channel will drive the SYNC pattern. Every microsecond gives one additional SYNC pulse beyond the hard coded 26 SYNC pulses.

The value is the number of microseconds times 2.

Default is 5 decimal which gives 10us of precharge which is 10 extra SYNC pulses for a total of 36.

Example:

For 10us precharge, program 5 (10us/2us).

15 — **Reserved**

Project: All

14 — **Reserved**

Project: All

13 — **Reserved**

Project: All

12 — **Reserved**

Project: All

DP_AUX_CTL	
11	**Reserved** Project: All
10:0	**2X_Bit_Clock_divider** Project: All Format: Used to determine the 2X bit clock the Aux Channel logic runs on. This value divides the input clock frequency down to 2X bit clock rate. It should be programmed to get as close as possible to the ideal rate of 2MHz. The input clock is the rawclk. [DevCPT] rawclk=125mhz Example: For 125MHz input clock and desired 2MHz 2X bit clock, program 63 (125MHz/2MHz).

3.8.3 DP_AUX_DATA—DisplayPort AUX Channel Data

DP Aux Channel Data Format	
Project: All	
Bit	**Description**
31:0	**AUX_CH_DATA** Project: All A DWord of the message. Writes give the data to transmit during the transaction. The MSbyte is transmitted first. Reads will give the response data after transaction complete.

DP_AUX_DATA		
Register Type: MMIO **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 5x32		
The read value will not be valid while the DisplayPort Aux Channel Control Register Send/Busy bit is asserted.		
DWord	**Bit**	**Description**
0	31:0	**AUX_CH_DATA1** Project: All Format: DP Aux Channel Data Format
1	31:0	**AUX_CH_DATA2** Project: All Format: DP Aux Channel Data Format
2	31:0	**AUX_CH_DATA3** Project: All Format: DP Aux Channel Data Format
3	31:0	**AUX_CH_DATA4** Project: All Format: DP Aux Channel Data Format
4	31:0	**AUX_CH_DATA5** Project: All Format: DP Aux Channel Data Format

3.8.4 DP_BUFTRANS—DisplayPort Buffer Translation

DisplayPort Buffer Translation Format					
Project: All **Default Value:** 00000000h					
Bit	**Description**				
31:28	**Reserved**	Project:	All	Format:	MBZ
27:19	**OE** These bits select the OE vswing level	Project:	All	Range:	0..511
18:17	**Reserved**	Project:	All	Format:	MBZ
16:12	**Pre_Emphasis** These bits select the pre-emphasis level	Project:	All	Range:	0..31
11:10	**Reserved**	Project:	All	Format:	MBZ
9:6	**P_current_drive** These bits select the P current drive value	Project:	All	Range:	0..15
5:4	**Reserved**	Project:	All	Format:	MBZ
3:0	**N_current_drive** These bits select the N current drive value	Project:	All	Range:	0..15

Programming Requirements:

DP mode		Offset	Value
L1	0dB	0xE4F00	**0x0100030C**
L1	3.5dB	0xE4F04	**0x00B8230C**
L1	6dB	0xE4F08	**0x06F8930C**
L1	9.5dB	0xE4F0C	**0x05F8E38E**
L2	0dB	0xE4F10	**0x00B8030C**
L2	3.5dB	0xE4F14	**0x0B78830C**
L2	6dB	0xE4F18	**0x09F8D3CF**
L3	0dB	0xE4F1C	**0x01E8030C**
L3	3.5dB	0xE4F20	**0x09F863CF**
L4	0 dB	0xE4F24	**0x0FF803CF**

Vswing	0dB pre-emphasis	3.5dB pre-emphasis	6dB pre-emphasis	9.5dB pre-emphasis
400mV	DWord 0	DWord 1	DWord 2	DWord 3
600mV	DWord 4	DWord 5	Dword 6	Not supported
800mV	Dword 7	Dword 8	Not supported	Not supported
1200mV	Dword 9	Not supported	Not supported	Not supported

DP_BUFTRANS

Register Type: MMIO
Project: All
Default Value: 0100038Eh; 00B8338Eh; 0178838Eh; 09F8E38Eh; 00B8038Eh; 0978838Eh; 09F8B38E; 0178038Eh; 09F8638Eh; 09F8038Eh
Access: R/W
Size (in bits): 10x32

These registers define current drive, pre-emphasis and voltage swing buffer programming required for the different voltage swing and pre-emphasis settings in the DisplayPort Control.

The default value is not the optimal value. See the programming requirements above for the correct values to use.

DWord	Bit	Description
0	31:0	**Vswing400mV_Pre0_0dB** Project: All Format: DisplayPort Buffer Translation Format See Description Above
1	31:0	**Vswing400mV_Pre3_5dB** Project: All Format: DisplayPort Buffer Translation Format See Description Above
2	31:0	**Vswing400mV_Pre6_0dB** Project: All Format: DisplayPort Buffer Translation Format See Description Above
3	31:0	**Vswing400mV_Pre9_5dB** Project: All Format: DisplayPort Buffer Translation Format See Description Above
4	31:0	**Vswing600mV_Pre0_0dB** Project: All Format: DisplayPort Buffer Translation Format See Description Above
5	31:0	**Vswing600mV_Pre3_5dB** Project: All Format: DisplayPort Buffer Translation Format See Description Above
6	31:0	**Vswing600mV_Pre6_0dB** Project: All Format: DisplayPort Buffer Translation Format See Description Above
7	31:0	**Vswing800mV_Pre0_0dB** Project: All Format: DisplayPort Buffer Translation Format See Description Above

DP_BUFTRANS				
8	31:0	**Vswing800mV_Pre3_5dB**	Project:	All
		Format: DisplayPort Buffer Translation Format	See Description Above	
9	31:0	**Vswing1200mV_Pre0_0dB**	Project:	All
		Format: DisplayPort Buffer Translation Format	See Description Above	

4. South Display Engine Audio

4.1 Audio Programming Sequence

The following HDMI and DisplayPort audio programming sequences are for use when enabling or disabling audio or temporarily disabling audio during a display mode set.

The audio codec and audio controller disable sequences must be followed prior to disabling the transcoder or port in a display mode set.

The audio codec and controller enable sequences can be followed after the transcoder is enabled and the port is enabled and completed link training (not sending TP1, TP2, or Idle).

The audio controller and audio codec sequences may be done in parallel or serial. In general, the change in ELDV/PD in the codec sequence will generate an unsolicited response to the audio controller driver to indicate that the controller sequence should start, but other mechanisms may be used.

Audio codec disable sequence:

- Disable sample fabrication
    - Set AUD_MISC_CTRL Sample_Fabrication_EN (bit 2) to “0”.
- Disable timestamps
    - Set AUD_CONFIG N_value_index (bit 29) to "0" for HDMI or "1" for DisplayPort.
    - Set N_programming_enable (bit 28) to "1"
    - Set Upper_N_value and Lower_N_value (bits 28:20, 15:4) to all "0"s.
- Disable ELDV and ELD buffer
    - Set AUD_CNTRL_ST2 ELD_valid (bit 0, 4, or 8 based on which port is used) to "0"
- Wait for 2 vertical blanks
- Optional: Disable audio PD (Presence Detect)
    - Software may choose to skip this in order to keep PD enabled during a resolution switch.
    - Set the port control register (HDMI_CTL or DP_CTL) Audio_Output_Enable (bit 6) to "0".

Audio controller disable sequence:

- Program Stream ID to 0 – Verb ID 706
- Disable audio info frames transmission – Verb ID 732
- Disable Digen – Verb ID 70D
- Program the codec to D3 state if needed.
- Audio driver may stop the audio controller DMA engine at this point if needed, but not required.

Audio codec enable sequence:

- Enable audio Presence Detect
    - Set the port control register (HDMI_CTL or DP_CTL) Audio_Output_Enable (bit 6) to "1".
- Wait for 1 vertical blank
- Load ELD buffer and Enable ELDV

    - Set AUD_CNTRL_ST2 ELD_valid (bit 0, 4, or 8 based on which port is used) to "1".
- Enable timestamps
    - Set AUD_CONFIG N_value_index (bit 29) to "0" for HDMI or "1" for DisplayPort.
    - Set N_programming_enable (bit 28) to "0".
    - Program Upper_N_value and Lower_N_value (bits 28:20, 15:4) if a non-default N value is needed.
- Enable sample fabrication if this feature is needed
    - Set AUD_MISC_CTRL Sample_Fabrication_EN (bit 2) to “1”.

Audio controller enable sequence:

- Program the codec to D0 state if in D3 state.
- Program Stream ID to non zero – Verb ID 706
- Enable audio info frames transmission – Verb ID 732
- Enable Digen – Verb ID 70D
- If audio controller DMA engine is stopped, audio driver can start the DMA engine at this point.

4.2 Audio Configuration

4.2.1 AUD_CONFIG—Audio Configuration

AUD_CONFIG

Register Type: MMIO
Project: All
Default Value: 00000000h
Access: R/W
Size (in bits): 32

This register configures the audio output.

Bit	Description
31:30	**Reserved** Project: All Format:
29	**N_value_Index** Project: All Default Value: 0b

Value	Name	Description	Project
0b	HDMI	N value read on bits 27:20 and 15:4 reflects HDMI N value. Bits 27:20 and 15:4 are programmable to any N value – default h7FA6.	All
1b	Display Port	N value read on bits 27:20 and 15:4 reflects DisplayPort N value. Set this bit to 1 before programming N value register. When this bit is set to 1, 27:20 and 15:4 will reflect the current N value – default h8000.	All

28	**Reserved** Project: All

AUD_CONFIG

27:20	**Reserved** Project: All
19:16	**Pixel_Clock_HDMI** Project: All Default Value: 0b This is the target frequency of the CEA/HDMI video mode to which the audio stream is added. This value is used for generating N_CTS packets. **This refers to only HDMI Pixel clock and does not refer to DisplayPort Link clock. DisplayPort Link clock does not require this programming.** **Note:** The Transcoder on which audio is attached must be disabled when changing this field.
15:4	**Reserved** Project: All
3	**Disable_NCTS** Project: All Set this bit to disable N & CTS or M generation for CTM modes. This is to enable prediction of CRC in CTM modes.
2:0	**Reserved** Project: All Format:

Value	Name	Description	Project
0000b	25.2 / 1.001 MHz	25.2 / 1.001 MHz	All
0001b	25.2 MHz	25.2 MHz (Program this value for pixel clocks not listed in this field)	All
0010b	27 MHz	27 MHz	All
0011b	27 * 1.001 MHz	27 * 1.001 MHz	All
0100b	54 MHz	54 MHz	All
0101b	54 * 1.001 MHz	54 * 1.001 MHz	All
0110b	74.25 / 1.001 MHz	74.25 / 1.001 MHz	All
0111b	74.25 MHz	74.25 MHz	All
1000b	148.5 / 1.001 MHz	148.5 / 1.001 MHz	All
1001b	148.5 MHz	148.5 MHz	All
Others	Reserved	Reserved	All

4.2.2 AUD_CTS_ENABLE – Audio CTS Programming Enable

AUD_CTS_ENABLE	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32

These values are returned from the device as the Subordinate Node Count response to a Get Root Node command.

Bit	Description
31:22	**Reserved** Project: All Format:
21	**CTS_M value Index** Project: All Default Value: 0b

Value	Name	Description	Project
0b	CTS	CTS value read on bits 23:4 reflects CTS value. Bit 23:4 is programmable to any CTS value. default is 0	All
1b	M	M value read on bits 21:4 reflects DisplayPort M value. Set this bit to 1 before programming M value register. When this is set to 1 23:4 will reflect the current N value	All

20	**Enable_CTS_or_M_programming** Project: All When set will enable CTS or M programming.
19:0	**CTS_programming** Project: All These are bits [19:0] of programmable CTS values for non-CEA modes. Bit 21 of this register must also be written in order to enable programming. Please note that the Transcoder to which audio is attached must be disabled when changing this field.

4.2.3 AUD_MISC_CTRL—Audio MISC Control

AUD_MISC_CTRL	
Register Type:	MMIO
Project:	All
Default Value:	00000044h
Access:	Read Only
Size (in bits):	32

Bit	Description
31:9	**Reserved** Project: All Format: MBZ
8	**Reserved** Project: All
7:4	**Output_Delay** Project: All Default Value: 0100b The number of samples between when the sample is received from the HD Audio link and when it appears as an analog signal at the pin.
3	**Reserved** Project: All Format: MBZ
2	**Sample_Fabrication_EN_bit** Project: All Access: R/W Default Value: 0b This bit indicates whether internal fabrication of audio samples is enabled during a link underrun.

Value	Name	Description	Project
0b	Disable	Audio fabrication disabled	All
1b	Enable	Audio fabrication enabled	All

Bit	Description
1	**Pro_Allowed** Project: All Access: R/W Default Value: 0b By default, the audio device is configured to consumer mode and does not allow the mode to be changed to professional mode by an HD Audio verb. When Pro is allowed by setting this configuration bit, the HD Audio codec allows a verb to set the device into professional mode. Note: Setting this configuration bit does not change the default Pro bit value to be 1. Pro must be set to 1 through the normal process, using a verb.

Value	Name	Description	Project
0b	Consumer	Consumer use only	All
1b	Professional	Professional use allowed	All

Bit	Description
0	**Reserved** Project: All Format: MBZ

4.2.4 AUD_VID_DID—Audio Vendor ID / Device ID

AUD_VID_DID	
Register Type:	MMIO
Project:	All
Default Value:	80862805h
Access:	Read Only
Size (in bits):	32

These values are returned from the device as the Vendor ID/ Device ID response to a Get Root Node command.

Bit	Description
31:16	**Vendor_ID** Project: All Format: Used to identify the codec within the PnP system. This field is hardwired within the device. Value = 0x8086
15:0	**Device_ID** Project: All Format: Constant used to identify the codec within the PnP system. This field is set by the device hardware. Value = 0x2805 [Cougarpoint]

4.2.5 AUD_RID—Audio Revision ID

AUD_RID	
Register Type:	MMIO
Project:	All
Default Value:	00100000h
Access:	Read Only
Size (in bits):	32

These values are returned from the device as the Revision ID response to a Get Root Node command.

Bit	Description
31:24	**Reserved** Project: All Format:
23:20	**Major_Revision** Project: All Default Value: 0001b The major revision number (left of the decimal) of the HD Audio Spec to which the codec is fully compliant. This field is hardwired within the device. Value = 0x1
19:16	**Minor_Revision** Project: All The minor revision number (rights of the decimal) or “dot number” of the HD Audio Spec to which the codec is fully compliant. This field is hardwired within the device. Value = 0x0
15:8	**Revision_ID** Project: All The vendor’s revision number for this given Device ID. This field is hardwired within the device. Value = 0x0

AUD_RID	
7:0	**Stepping_ID** Project: All An optional vendor stepping number within the given Revision ID. This field is hardwired within the device. Value = 0x0

4.2.6 AUD_PWRST—Audio Power State

Audio Power State Format	
Project: All	
Bit	**Description**
1:0	**Power_State** Project: All Default Value: 11b D3

Value	Name	Description	Project
00b	D0	D0	All
01b,10b	Unsupported	Unsupported	All
11b	D3	D3	All

AUD_PWRST	
Register Type: MMIO **Project:** All **Default Value:** 00FFFFFFh **Access:** Read Only **Size (in bits):** 32	
These values are returned from the device as the Power State response to a Get Audio Function Group command.	
Bit	**Description**
31:28	**Reserved** Project: All Format:
27:26	**Func_Grp_Dev_PwrSt_Curr** Project: All Format: Audio Power State Format Function Group Device current power state
25:24	**Func_Grp_Dev_PwrSt_Set** Project: All Format: Audio Power State Format Function Group Device power state that was set
23:22	**ConvC_Widget_PwrSt_Curr** Project: All Format: Audio Power State Format ConverorC Widget current power state

AUD_PWRST					
21:20	**ConvC_Widget_PwrSt_Req** ConverorC Widget power state that was requested by audio software	Project:	All	Format:	Audio Power State Format
19:18	**ConvertorB_Widget_PwrSt_Curr** ConverorB Widget current power state	Project:	All	Format:	Audio Power State Format
17:16	**ConvertorB_Widget_PwrSt_Req** ConverorB Widget power state that was requested by audio software	Project:	All	Format:	Audio Power State Format
15:14	**ConvertorA_Widget_PwrSt_Curr** ConverorA Widget current power state	Project:	All	Format:	Audio Power State Format
13:12	**ConvertorA_Widget_PwrSt_Req** ConverorA Widget power state that was requested by audio software	Project:	All	Format:	Audio Power State Format
11:10	**PinD_Widget_PwrSt_Curr** PinD Widget current power state	Project:	All	Format:	Audio Power State Format
9:8	**PinD_Widget_PwrSt_Set** PinD Widget power state that was set	Project:	All	Format:	Audio Power State Format
7:6	**PinC_Widget_PwrSt_Curr** PinC Widget current power state	Project:	All	Format:	Audio Power State Format
5:4	**PinC_Widget_PwrSt_Set** PinC Widget power state that was set	Project:	All	Format:	Audio Power State Format
3:2	**PinB_Widget_PwrSt_Curr** PinB Widget current power state	Project:	All	Format:	Audio Power State Format
1:0	**PinB_Widget_PwrSt_Set** PinB Widget power state that was set	Project:	All	Format:	Audio Power State Format

4.2.7 AUD_PINW_CONNLNG_LIST—Audio Connection List

AUD_PINW_CONNLNG_LIST	
Register Type:	MMIO
Project:	All
Default Value:	00000302h
Access:	Read Only
Size (in bits):	32

These values are returned from the device as the Connection List Length response to a Get Pin Widget command.

Bit	Description
31:16	**Reserved** Project: All Format:
15:8	**Connection_List_Entry** Project: All Default Value: 03h Connection to Convertor Widget Node 0x03
7	**Long_Form** Project: All Default Value: 0b This bit indicates whether the items in the connection list are 'long form' or 'short form'. This bit is hardwired to 0 (items in connection list are short form)
6:0	**Connection_List_Length** Project: All Default Value: 02h This field indicates the number of items in the connection list. If this field is 2, there is only one hardwired input possible, which is read from the Connection List, and there is no Connection Select Control.

4.2.8 AUD_PINW_CONNLNG_SEL—Audio Connection Select

AUD_PINW_CONNLNG_SEL	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32

These values are returned from the device as the Connection List Length response to a Get Pin Widget command.

Bit	Description
31:24	**Reserved** Project: All Format:
23:16	**Connection_select_Control_D** Project: All Format: Connection Index Currently Set [Default 0x00], Port D Widget is set to 0x00
15:8	**Connection_select_Control_C** Project: All Format: Connection Index Currently Set [Default 0x00], Port C Widget is set to 0x00

AUD_PINW_CONNLNG_SEL	
7:0	**Connection_select_Control_B** Project: All Format: Connection Index Currently Set [Default 0x00], Port B Widget is set to 0x00

4.2.9 AUD_CNTL_ST—Audio Control State

AUD_CNTL_ST	
Register Type:	MMIO
Project:	All
Default Value:	00005400h
Access:	R/W
Size (in bits):	32
Bit	**Description**
31	**Reserved** Project: All Format: MBZ
30:29	**DIP_Port_Select** Project: All Access: Read Only Default Value: 00b This read-only bit reflects which port is used to transmit the DIP data. This can only change when DIP is disabled. If one or more audio-related DIP packets is enabled and audio is enabled on a digital port, these bits will reflect the digital port to which audio is directed.

Value	Name	Description	Project
00b	Reserved	Reserved	All
01b	Digital Port B	Digital Port B	All
10b	Digital Port C	Digital Port C	All
11b	Digital Port D	Digital Port D	All

28:25	**Reserved** Project: All Format: MBZ

AUD_CNTL_ST

24:21

DIP_type_enable_status

Project: All

Access: Read Only

Default Value: 0000b

These bits reflect the DIP types enabled. It can be updated while the port is enabled. Within 2 vblank periods, the DIP is guaranteed to have been transmitted. Disabling a DIP type results in setting the contents of that DIP buffer to zero. A reserved setting reflects a disabled DIP.

Value	Name	Description	Project
XXX0b	Disable	Audio DIP disabled	All
XXX1b	Enable	Audio DIP enabled	All
XX0Xb	Disable	Generic 1 (ACP) DIP disabled	All
XX1Xb	Enable	Generic 1 (ACP) DIP enabled	All
X0XXb	Disable	Generic 2 DIP disabled	All
X1XXb	Enable	Generic 2 DIP enabled, can be used by ISRC1 or ISRC2	All
1XXXb	Reserved	Reserved	All

20:18

DIP_buffer_index

Project: All

Default Value: 0000b

This field is used during read of different DIPs, and during read or write of ELD data. These bits are used as an index to their respective DIP or ELD buffers. When the index is not valid, the contents of the DIP will return all 0's.

Value	Name	Description	Project
000b	Audio	Audio DIP (31 bytes of address space, 31 bytes of data)	All
001b	Gen 1	Generic 1 (ACP) Data Island Packet (31 bytes of address space, 31 bytes of data)	All
010b	Gen 2	Generic 2 (ISRC1) Data Island Packet (31 bytes of address space, 31 bytes of data)	All
011b	Gen 3	Generic 3 (ISRC2) Data Island Packet (31 bytes of address space, 31 bytes of data)	All
Others	Reserved	Reserved	All

AUD_CNTL_ST

17:16 — DIP_transmission_frequency

Project: All

Access: Read Only

Default Value: 00b

These bits reflect the frequency of DIP transmission for the DIP buffer type designated in bits 20:18. When writing DIP data, this value is also latched when the first DW of the DIP is written.

When read, this value reflects the DIP transmission frequency for the DIP buffer designated in bits 20:18.

Value	Name	Description	Project
00b	Disable	Disabled	All
01b	Reserved	Reserved	All
10b	Send Once	Send Once	All
11b	Best Effort	Best effort (Send at least every other vsync)	All

15 — Reserved Project: All Format: MBZ

14:10 — ELD_buffer_size Project: All Access: Read Only

10101 = This field reflects the size of the ELD buffer in DWORDs (84 Bytes of ELD)

9:5 — ELD_access_address Project: All

Selects the DWORD address for access to the ELD buffer (84 bytes). The value wraps back to zero when incremented past the max addressing value 0x1F. This field change takes effect immediately after being written. The read value indicates the current access address.

4 — ELD_ACK Project: All

Acknowledgement from the audio driver that ELD read has been completed

3:0 — DIP_access_address Project: All

Selects the DWORD address for access to the DIP buffers. The value wraps back to zero when it incremented past the max addressing value of 0xF. This field change takes effect immediately after being written. The read value indicates the current access address.

4.2.10 AUD_CNTRL_ST2— Audio Control State 2

AUD_CNTRL_ST2	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

This register is used for handshaking between the audio and video drivers for interrupt management. For each port, ELD and content protection readiness is sent by the display software to the audio software via an unsolicited response when the ELD or CP ready bit is set. Display software sets these bits as part of enabling the respective audio-enabled digital port.

Bit	Description
31:10	**Reserved** Project: All Format:
9	**CP_ReadyD** Project: All Default Value: 0b This **R/W** bit reflects the state of CP request from the audio unit. When an audio CP request has been serviced, it must be reset to 1 by the video software to indicate that the CP request has been serviced.

Value	Name	Description	Project
0b	Pending or Not Ready	CP request pending or not ready to receive requests	All
1b	Ready	CP request ready	All

Bit	Description
8	**ELD_validD** Project: All Default Value: 0b This **R/W** bit reflects the state of the ELD data written to the ELD RAM. After writing the ELD data, the video software must set this bit to 1 to indicate that the ELD data is valid. At audio codec initialization, or on a hotplug event, this bit is set to 0 by the video software. This bit is reflected in the audio pin complex widget as the ELD valid status bit.

Value	Name	Description	Project
0b	Invalid	ELD data invalid (default, when writing ELD data, set 0 by software)	All
1b	Valid	ELD data valid (Set by video software only)	All

Bit	Description
7:6	**Reserved** Project: All Format:

AUD_CNTRL_ST2

5	**CP_ReadyC** Project: All Default Value: 0b See CP_ReadyD description.

Value	Name	Description	Project
0b	Not Ready	CP request pending or not ready to receive requests	All
1b	Ready	CP request ready	All

4	**ELD_validC** Project: All Default Value: 0b See ELD_validD descripion.

Value	Name	Description	Project
0b	Invalid	ELD data invalid (default, when writing ELD data, set 0 by software)	All
1b	Valid	ELD data valid (Set by video software only)	All

3:2	**Reserved** Project: All Format:
1	**CP_ReadyB** Project: All Default Value: 0b See CP_ReadyD description.

Value	Name	Description	Project
0b	Not Ready	CP request pending or not ready to receive requests	All
1b	Ready	CP request ready	All

0	**ELD_validB** Project: All Default Value: 0b See ELD_validD descripion.

Value	Name	Description	Project
0b	Invalid	ELD data invalid (default, when writing ELD data, set 0 by software)	All
1b	Valid	ELD data valid (Set by video software only)	All

4.2.11 AUD_HDMIW_HDMIEDID—Audio HDMI Data EDID Block

AUD_HDMIW_HDMIEDID	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

These registers contain the HDMI data block from the EDID. The graphics driver reads the EDID and writes the structure to these registers. The vendor specific data block may be longer than 8 bytes, but the driver must not write more than 48 bytes to the buffer. The EDID format is Version 3 within the CEA-861B specification. The HDMI Vendor Specific Data Block is described in version 1.1 of the HDMI specification.

These values are returned from the device as the HDMI Vendor Specific Data Block response to a Get HDMI Widget command.

Writing sequence:

- Video software sets ELD invalid, and sets the ELD access address to 0, or to the desired DWORD to be written.
- Video software writes ELD data 1 DWORD at a time. The ELD access address autoincrements with each DWORD write, wrapping around to address 0 when the max buffer address size of 0xF has been reached. Please note that software must write an entire DWORD at a time.
- Please note that the audio driver checks the valid bit with each byte read of the ELD. This means that the video driver can unilaterally write ELD irrespective of audio driver ELD read status.

Reading sequence:

- Video software sets the ELD access address to 0, or to the desired DWORD to be read.
- Video software reads ELD data 1 DWORD at a time. The ELD access address autoincrements with each DWORD read, wrapping around to address 0 when the max buffer address size of 0xF has been reached.

Bit	Description
31:0	**EDID_HDMI_Data_Block** Project: All Format: Please note that the contents of this buffer are not cleared when ELD is disabled. The contents of this buffer are cleared during gfx reset

4.2.12 AUD_HDMIW_INFOFR—Audio Widget Data Island Packet

AUD_HDMIW_INFOFR	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access:	Read Only
Size (in bits):	32

When the IF type or dword index is not valid, the contents of the DIP will return all 0's.

These values are programmed by the audio driver in an HDMI Widget Set command. They are returned one byte at a time from the device on the HD audio bus as the HDMI DIP response to a Get HDMI Widget command. To fetch a specific byte, the audio driver should send an HDMI Widget HDMI DIP Index Pointer Set command to set the index, then fetch the indexed byte using the HDMI DIP get.

Bit	Description
31:0	**Data_Island_Packet_Data** Project: All Format: This reflects the contents of the DIP indexed by the DIP access address. The contents of this buffer are cleared during function reset or HD audio link reset.

5. South Display Engine Transcoder and FDI Control

5.1 Transcoder Control

5.1.1 TRANS_CONF—Transcoder Configuration

TRANS_CONF	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank OR transcoder disabled

Bit	Description
31	**Transcoder_Enable** Project: All Setting this bit to the value of one, turns on the transcoder. Turning the transcoder off disables the timing generator and synchronization pulses to the display will not be maintained. Transcoder timing registers must contain valid values before this bit is enabled.

Value	Name	Description	Project
0b	Disable	Disabled	All
1b	Enable	Enabled	All

Bit	Description
30	**Transcoder_State** Project: All This **read only** bit indicates the actual state of the transcoder. Since there can be some delay between disabling the transcoder and the transcoder actually shutting off, this bit indicates the true current state of the transcoder.

Value	Name	Description	Project
0b	Disabled	Transcoder is disabled	All
1b	Enabled	Transcoder is enabled	All

Bit	Description
29:24	**Reserved** Project: All Format:

TRANS_CONF

23:21	**Interlaced_Mode** Project: All Default Value: 000b These bits are used for control of the transcoder interlaced mode.

Value	Name	Description	Project
000b	Progressive	Progressive	All
011b	Interlaced	Interlaced (north display must also be set to interlaced)	All
Others	Reserved	Reserved	All

20:11	**Reserved** Project: All Format:
10	**xvYCC_Color_Range_Limit** Project: All Default Value: 0b This bit is used to limit the color range of the port outputs from 1 to 254 for 8-bit components, 4 to 109 for 10-bit components, or 16 to 4079 for 12-bit components. Values outside of the range will be clamped to fit within the range. There is no need to set this bit if the equivalent bit is set in the north display pipe configuration register.

Value	Name	Description	Project
0b	Full	Do not limit the range	All
1b	Limit	Limit range	All

9:0	**Reserved** Project: All Format:

5.2 FDI Receiver

5.2.1 FDI_RX_CTL— FDI Rx Control

FDI_RX_CTL	
Register Type:	MMIO
Project:	All
Default Value:	00000040h
Access:	R/W
Size (in bits):	32
Double Buffer Update Point:	Depends on bit

Bit	Description
31	**FDI_Rx_Enable** Project: All Default Value: 0b Disabling this port will put it in its lowest power state. Port enable takes place on the Vblank after being written.

Value	Name	Description	Project
0b	Disable	Disable and tristate the FDI Rx interface	All
1b	Enable	Enable the FDI Rx interface	All

30:28	**Reserved** Project: All Format: MBZ
27	**FS_error_correction_enable** Project: All Default Value: 0b This bit enables the Fill Start error correction over FDI. Once the FS code is incorrectly received, the receiver will recover the FS code. The FDI Rx TU size register must be set correctly. The following condition must be met in order to set this bit: **(Active+2)/TU >= 1**

Value	Name	Description	Project
0b	Disable	Disable FS Error Correction	All
1b	Enable	Enable FS Error Correction	All

FDI_RX_CTL

26 — **FE_error_correction_enable**

Project: All

Default Value: 0b

This bit enables the Fill End error correction over FDI. Once the FE code is incorrectly received, the receiver will recover the FE code. The FDI Rx TU size register must be set correctly. The following condition must be met in order to set this bit: **(Active+2)/TU >= 1**

Value	Name	Description	Project
0b	Disable	Disable FE Error Correction	All
1b	Enable	Enable FE Error Correction	All

25 — **FS_error_reporting_enable**

Project: All

Default Value: 0b

This bit enables the FS error reporting over FDI. Once the FS code is incorrectly received, the receiver will report the FS code error through an interrupt. The FDI Rx TU size register must be set correctly. The following condition must be met in order to set this bit: **(Active+2)/TU >= 1**

Value	Name	Description	Project
0b	Disable	Disable FS Error Reporting	All
1b	Enable	Enable FS Error Reporting	All

24 — **FE_error_reporting_enable**

Project: All

Default Value: 0b

This bit enables the FE error reporting over FDI. Once the FE code is incorrectly received, the receiver will report the FE code error through an interrupt. The FDI Rx TU size register must be set correctly. The following condition must be met in order to set this bit: **(Active+2)/TU >= 1**

Value	Name	Description	Project
0b	Disable	Disable FE Error Reporting	All
1b	Enable	Enable FE Error Reporting	All

23:22 — **Reserved** Project: All Format:

FDI_RX_CTL

21:19

Port_Width_Selection

Project: All

Default Value: 0b

These bits select the number of lanes to be enabled on the link. Port width change must be done as a part of mode set.

[DevCPT] FDI B and FDI C share lanes. FDI C maximum port width is 2 lanes. FDI B maximum port width is 4 lanes when FDI C is disabled, 2 lanes when FDI C is enabled.

Locked once port is enabled. Updates when the port is disabled then re-enabled

Value	Name	Description	Project
000b	x1 Mode	x1 Mode	All
001b	x2 Mode	x2 Mode	All
010b	x3 Mode	x3 Mode	All
011b	x4 Mode	x4 Mode	All
Others	Reserved	Reserved	All

18:16

Bits_Per_Color

Project: All

Default Value: 0b

This field selects the number of bits per color sent over the link. Color format takes place on the Vblank after being written.

Value	Name	Description	Project
000b	8 bpc	8 bits per color	All
001b	10 bpc	10 bits per color	All
010b	6 bpc	6 bits per color	All
011b	12 bpc	12 bits per color	All
Others	Reserved	Reserved	All

15

Link_reversal_strap_override

Project: All

Default Value: 0b

Link is reversed if DMI is reversed. This bit overrides the status of DMI reversal to the reverse of what is strapped. It must be set before the link is enabled in order to take effect. Writing to this bit when the link is enabled has no effect. All FDI links must be off in order for this bit to take effect. This bit is ORed with the link reversal strap overrides from any other FDI Rx Control registers.

Value	Name	Description	Project
0b	Not Overwritten	Link reversal strap not overwritten	All
1b	Overwritten	Link reversal strap overwritten.	All

FDI_RX_CTL

14 — **DMI_Link_reversal_status**

Project: All

Access: Read Only

Default Value: 0b

This bit reflects the DMI link reversal strap.

Value	Name	Description	Project
0b	Not Reversed	Link not reversed	All
1b	Reversed	Link reversed.	All

13 — **FDI_PLL_enable** Project: All Format: Enable

This bit enables the FDI PLL. After enabling the FDI PLL, software must wait for a warmup period after before enabling the link. This bit is ORed with the FDI PLL enables from any other FDI Rx Control registers.

12 — **Reserved** Project: All Format:

11 — **Composite_Sync_Select**

Project: All

Default Value: 0b

This bit selects between composite Sync and separate Fsync/Lsync on this port. Composite sync must be used when three pipes are enabled.

Value	Name	Description	Project
0b	Separate	Separate Fsync/Lsync	All
1b	Composite	Composite Sync	All

10 — **FDI_Auto_Train**

Project: All

Default Value: 0b

This bit enables FDI auto-training on this port. Locked once port is enabled. Updates when port is disabled.

Value	Name	Description	Project
0b	Disable	Disable FDI auto-training	All
1b	Enable	Enable FDI auto-training	All

FDI_RX_CTL

9:8 Link_training_pattern_enable

Project: All

Default Value: 0b

These bits are used for link initialization. Please note that the link must first be configured prior to sending training patterns

When enabling the port, it must be turned on with pattern 1 enabled. When retraining, the port must be disabled, then re-enabled with pattern 1 enabled.

Value	Name	Description	Project
00b	Pattern 1	Pattern 1 enabled: Repetition of D10.2 characters.	All
01b	Pattern 2	Pattern 2 enabled: Repetition of K28.5, D11.6, K28.5, D11.6, D10.2, D10.2, D10.2, D10.2, D10.2, D10.2. Please note that the entire pattern must complete before another pattern is sent. Scrambling initialization and disparity init commence at the end of the last iteration of pattern 2.	All
10b	Idle	Idle Pattern enabled: Transmit BS followed by VB-ID with NoVideoStream_flag set to 1, five times	All
11b	Normal	Link not in training: Send normal pixels	All

7 Reserved

Project: All

6 Enhanced_Framing_Enable

Project: All

Default Value: 1b Enhanced framing enabled

This bit selects enhanced framing.

Locked once port is enabled. Updates when the port is disabled then re-enabled

Value	Name	Description	Project
0b	Disable	Enhanced framing disabled	All
1b	Enable	Enhanced framing enabled	All

5 Reserved

Project: All

FDI_RX_CTL	
4	**Rawclk_to_PCDCLK_selection** Project: All Default Value: 0b This bit switches PCH display clocking between the raw oscillator clock ([DevCPT] rawclk=125mhz)and PCDCLK ([DevCPT] PCDCLK=450mhz). It must be programmed as part of enabling and disabling the link. Please see the mode set / boot flow for more detail. This bit is ORed with the selection bit from any other FDI Rx Control registers.

Value	Name	Description	Project
0b	Rawclk	Rawclk used	All
1b	PCDCLK	PCDCLK used	All

3:0	**Reserved** Project: All Format: MBZ

5.2.2 FDI_RX_MISC— FDI Rx Miscellaneous

FDI_RX_MISC	
Register Type:	MMIO
Project:	All
Default Value:	00000080h
Access:	R/W
Size (in bits):	32
Bit	**Description**
31:22	**Reserved** Project: All Format: MBZ
21:20	**TP1_to_TP2_Time** Project: All Default Value: 0b These bits select the number of link clocks to count before transitioning from TP1 to TP2 during auto training. Program to 48 clocks before enabling FDI with auto-training.

Value	Name	Description	Project
10b	48	48 clocks - **required programming**	All
11b	64	64 clocks	All

19	**Reserved** Project: All Format: MBZ

FDI_RX_MISC	
18:16	**Bit_Lock_Timeout_Time** Project: All Default Value: 0b These bits select the number of link clocks to count before timing out on bit lock during auto training.

Value	Name	Description	Project
000b	128	128 clocks	All
001b	256	256 clocks	All
010b	384	384 clocks	All
011b	512	512 clocks	All
100b	640	640 clocks	All
101b	768	768 clocks	All
110b	896	896 clocks	All
111b	1024	1024 clocks	All

15:13	**Reserved** Project: All Format: MBZ
12:0	**FDI_Delay** Project: All Default Value: 80h Default 80h. This field specifies latency as relative delay with respect to the dot clock required for active data over the FDI interface to reach the timing generator FIFO in the transcoder.

5.2.3 FDI_RX_IMR — FDI Rx Interrupt Mask

FDI Receiver Interrupt Bit Definition	
Project: All	
FDI Receiver (FDI Rx) interrupt bits come from FDI Receiver events. The FDI_RX_IIR bits are ORed together to generate the FDI_RX Combined Interrupt which will appear in the South Display Engine Interrupt Control Registers. The FDI Receiver Interrupt Control Registers all share the same bit definitions from this table.	
Bit	**Description**
31:12	**Reserved** Project: All Format:
11	**FDI_RX_Bit_Lock_Timeout** Project: All Format: This indicates that bit lock timeout occured.
10	**FDI_RX_Interlane_Alignment** Project: All Format: This indicates all the lanes are properly inter-lane aligned.
9	**FDI_RX_Symbol_Lock** Project: All Format: This indicates training pattern 2 was consecutively received successfully on all the enabled lanes.
8	**FDI_RX_Bit_Lock** Project: All Format: This indicates D10.2 pattern in training pattern 1 was consecutively received successfully on all the enabled lanes.

FDI Receiver Interrupt Bit Definition				
7	**FDI_RX_Training_Pattern_2_Fail** This indicates that the training pattern 2 has failed.	Project:	All	Format:
6	**FS_Code_Error** This reports the Fill Start code missing condition.	Project:	All	Format:
5	**FE_Code_Error** This reports the Fill End code missing condition.	Project:	All	Format:
4	**FDI_RX_High_Symbol_Error_Rate** This indicates the received symbol error rate is more than 10^-10.	Project:	All	Format:
3	**Reserved**	Project:	All	Format:
2	**FDI_RX_Pixel_FIFO_Overflow**	Project:	All	Format:
1	**FDI_RX_Cross_Clock_FIFO_Overflow** This indicates the cross clock symbol clock to display clock FIFO overflowed.	Project:	All	Format:
0	**FDI_RX_Symbol_Queue_overflow** This indicates the symbol queue overflowed.	Project:	All	Format:

FDI_RX_IMR

Register Type: MMIO
Project: All
Default Value: 000007FFh
Access: R/W
Size (in bits): 32

See the interrupt bit definition table to find the source event for each interrupt bit.

Bit	Description
31:0	**Interrupt_Mask_Bits** Project: All This field contains a bit mask which selects which FDI_RX events are reported int the FDI_RX_IIR.

Value	Name	Description	Project
0b	Not Masked	Not Masked – will be reported in the FDI_RX_IIR	All
1b	Masked	Masked – will not be reported in the FDI_RX_IIR	All

5.2.4 FDI_RX_IIR — FDI Rx Interrupt Identity

FDI_RX_IIR	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access:	R/W Clear
Size (in bits):	32

See the interrupt bit definition table to find the source event for each interrupt bit.

Bit	Description
31:0	**Interrupt_Identity_Bits** Project: All This field holds the persistent values of the FDI_RX interrupt bits which are unmasked by the FDI_RX_IMR. Bits set in this register will propagate to the combined FDI_RX interrupt in the SDE_ISR. Bits set in this register will remain set (persist) until the interrupt condition is cleared by writing a '1' to the appropriate bits.

Value	Name	Description	Project
0b	Condition Not Detected	Interrupt Condition Not Detected	All
1b	Condition Detected	Interrupt Condition Detected	All

5.2.5 FDI_RX_TUSIZE— FDI Rx Transfer Unit Size

The FDI Receiver TU1 and TU2 sizes must be programmed to match the TU sizes used by the FDI Transmitter.

When switching between two refresh rates, both the TU1 and TU2 values must be programmed. For dynamic refresh rate control, TU1 values are the primary values and are used for the normal setting, and TU2 values are the secondary values and are used for the lower power setting.

FDI_RX_TUSIZE	
Register Type:	MMIO
Project:	All
Default Value:	7E000000h
Access:	R/W
Size (in bits):	32

Bit	Description
31	**Reserved** Project: All Format: MBZ
30:25	**TU_Size** Project: All This field is the size of the transfer unit for FDI, minus one.
24:0	**Reserved** Project: All Format: MBZ

Revision History

Revision Number	Description	Revision Date
1.0	First 2011 OpenSource edition	May 2011

§§

Intel® OpenSource HD Graphics Programmer's Reference Manual (PRM)

Supplement 1 to SNB - Volume 3 Part 3: PCH Display Registers (SandyBridge)

For the 2011 Intel Core Processor Family

May 2016

Revision 1.0

NOTICE:

This document contains information on products in the design phase of development, and Intel reserves the right to add or remove product features at any time, with or without changes to this open source documentation

Creative Commons License

You are free to Share — to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

No Derivative Works. You may not alter, transform, or build upon this work.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The SandyBridge chipset family, Havendale/Auburndale chipset family, Intel® 965 Express Chipset Family, Intel® G35 Express Chipset, and Intel® 965GMx Chipset Mobile Family Graphics Controller may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel® sales office or your distributor to obtain the latest specifications and before placing your product order. I2C is a two-wire communications bus/protocol developed by Philips. SMBus is a subset of the I2C bus/protocol and was developed by Intel®. Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

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

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

Copyright © 2011, Intel Corporation. All rights reserved.

This document supplements the *Intel® OpenSource HD Graphics Programmer's Reference Manual (PRM), Volume 3 Part 3: PCH Display Registers (SandyBridge).*

Section 1.1.4 Register Instances and Address Offsets of the document refers to a missing spreadsheet that contains address offset values for PCH Registers. Below is this spreadsheet that lists all address offset values for all of the PCH registers.

Included are the start and stop address values, along with the Short Name, Format, and Full Name of each register. Please refer to this table when reading section 1.1.4. Subsequent project specifications do include address offset values within each PCH register.

Address Offset Values for SNB PCH Registers

Start Address	Stop Address	Short name	Format	Full Name
0xC4000	0xC4003	SDE_ISR	ISR	South DE Interrupt Status
0xC4004	0xC4007	SDE_IMR	IMR	South DE Interrupt Mask
0xC4008	0xC400B	SDE_IIR	IIR	South DE Interrupt Identity
0xC400C	0xC400F	SDE_IER	IER	South DE Interrupt Enable
0xC4030	0xC4033	SHOTPLUG_CTL	SHOTPLUG_CTL	South Hot Plug control
0xC5010	0xC5027	GPIO_CTL_[0-5]	GPIO_CTL_[0-5]	GPIO Control
0xC5100	0xC5103	GMBUS0	GMBUS0	GMBUS0 Clock/Port Select
0xC5104	0xC5107	GMBUS1	GMBUS1	GMBUS1 Command/Status
0xC5108	0xC510B	GMBUS2	GMBUS2	GMBUS2 Status
0xC510C	0xC510F	GMBUS3	GMBUS3	GMBUS3 Data Buffer
0xC5110	0xC5113	GMBUS4	GMBUS4	GMBUS4 Interrupt Mask
0xC5120	0xC5123	GMBUS5	GMBUS5	GMBUS5 2 Byte Index
0xC6014	0xC6017	DPLL_CTL_A	DPLL_CTL	DPLLA Control
0xC6018	0xC601B	DPLL_CTL_B	DPLL_CTL	DPLLB Control
0xC6040	0xC6043	DPLL_FP0_A	DPLL_FP0	DPLLA Divisor 0
0xC6044	0xC6047	DPLL_FP1_A	DPLL_FP1	DPLLA Divisor 1
0xC6048	0xC604B	DPLL_FP0_B	DPLL_FP0	DPLLB Divisor 0
0xC604C	0xC604F	DPLL_FP1_B	DPLL_FP1	DPLLB Divisor 1
0xC6200	0xC6203	DREF_CTL	DREF_CTL	Display Reference Clock Control
0xC6204	0xC6207	RAWCLK_FREQ	RAWCLK_FREQ	Rawclk Frequency
0xC6210	0xC6213	SSC4_PARMS	SSC4_PARMS	SSC4 Parameters
0xC7000	0xC7003	DPLL_SEL	DPLL_SEL	DPLL Select
0xC7200	0xC7203	PP_STATUS	PP_STATUS	Panel Power Status
0xC7204	0xC7207	PP_CONTROL	PP_CONTROL	Panel Power Control

Address Offset Values for SNB PCH Registers				
Start Address	**Stop Address**	**Short name**	**Format**	**Full Name**
0xC7208	0xC720B	PP_ON_DELAYS	PP_ON_DELAYS	Panel Power On Sequencing Delays
0xC720C	0xC720F	PP_OFF_DELAYS	PP_OFF_DELAYS	Panel Power Off Sequencing Delays
0xC7210	0xC7213	PP_DIVISOR	PP_DIVISOR	Panel Power Cycle Delay and Reference Divisor
0xC8250	0xC8253	SBLC_PWM_CTL1	SBLC_PWM_CTL1	South BLM Control 1
0xC8254	0xC8257	SBLC_PWM_CTL2	SBLC_PWM_CTL2	South BLM Control 2
0xE0000	0xE0003	TRANS_HTOTAL_A	HTOTAL	Transcoder A Horizontal Total
0xE0004	0xE0007	TRANS_HBLANK_A	HBLANK	Transcoder A Horizontal Blank
0xE0008	0xE000B	TRANS_HSYNC_A	HSYNC	Transcoder A Horizontal Sync
0xE000C	0xE000F	TRANS_VTOTAL_A	VTOTAL	Transcoder A Vertical Total
0xE0010	0xE0013	TRANS_VBLANK_A	VBLANK	Transcoder A Vertical Blank
0xE0014	0xE0017	TRANS_VSYNC_A	VSYNC	Transcoder A Vertical Sync
0xE0028	0xE002B	TRANS_VSYNCSHIFT_A	VSYNCSHIFT	Transcoder A Vertical Sync Shift
0xE0030	0xE0033	TRANS_DATAM1_A	DATAM	Transcoder A Data M value 1
0xE0034	0xE0037	TRANS_DATAN1_A	DATAN	Transcoder A Data N value 1
0xE0038	0xE003B	TRANS_DATAM2_A	DATAM	Transcoder A Data M value 2
0xE003C	0xE003F	TRANS_DATAN2_A	DATAN	Transcoder A Data N value 2
0xE0040	0xE0043	TRANS_LINKM1_A	LINKM	Transcoder A Link M value 1
0xE0044	0xE0047	TRANS_LINKN1_A	LINKN	Transcoder A Link N value 1
0xE0048	0xE004B	TRANS_LINKM2_A	LINKM	Transcoder A Link M value 2
0xE004C	0xE004F	TRANS_LINKN2_A	LINKN	Transcoder A Link N value 2
0xE0200	0xE0203	VIDEO_DIP_CTL_A	VIDEO_DIP_CTL	Transcoder A Video Data Island Packet Control
0xE0208	0xE020B	VIDEO_DIP_DATA_A	VIDEO_DIP_DATA	Transcoder A Video Data Island Packet Data

Address Offset Values for SNB PCH Registers				
Start Address	**Stop Address**	**Short name**	**Format**	**Full Name**
0xE0210	0xE0213	VIDEO_DIP_GCP_A	VIDEO_DIP_GCP	Transcoder A Video Data Island Payload
0xE0300	0xE0303	TRANS_DP_CTL_A	TRANS_DP_CTL	Transcoder A DisplayPort Control
0xE1000	0xE1003	TRANS_HTOTAL_B	HTOTAL	Transcoder B Horizontal Total
0xE1004	0xE1007	TRANS_HBLANK_B	HBLANK	Transcoder B Horizontal Blank
0xE1008	0xE100B	TRANS_HSYNC_B	HSYNC	Transcoder B Horizontal Sync
0xE100C	0xE100F	TRANS_VTOTAL_B	VTOTAL	Transcoder B Vertical Total
0xE1010	0xE1013	TRANS_VBLANK_B	VBLANK	Transcoder B Vertical Blank
0xE1014	0xE1017	TRANS_VSYNC_B	VSYNC	Transcoder B Vertical Sync
0xE1028	0xE102B	TRANS_VSYNCSHIFT_B	VSYNCSHIFT	Transcoder B Vertical Sync Shift
0xE1030	0xE1033	TRANS_DATAM1_B	DATAM	Transcoder B Data M value 1
0xE1034	0xE1037	TRANS_DATAN1_B	DATAN	Transcoder B Data N value 1
0xE1038	0xE103B	TRANS_DATAM2_B	DATAM	Transcoder B Data M value 2
0xE103C	0xE103F	TRANS_DATAN2_B	DATAN	Transcoder B Data N value 2
0xE1040	0xE1043	TRANS_LINKM1_B	LINKM	Transcoder B Link M value 1
0xE1044	0xE1047	TRANS_LINKN1_B	LINKN	Transcoder B Link N value 1
0xE1048	0xE104B	TRANS_LINKM2_B	LINKM	Transcoder B Link M value 2
0xE104C	0xE104F	TRANS_LINKN2_B	LINKN	Transcoder B Link N value 2
0xE1100	0xE1103	DAC_CTL	DAC_CTL	Analog Port CRT DAC Control
0xE1140	0xE1143	HDMI_CTL_B	HDMI_CTL	HDMI Port B Control
0xE1150	0xE1153	HDMI_CTL_C	HDMI_CTL	HDMI Port C Control
0xE1160	0xE1163	HDMI_CTL_D	HDMI_CTL	HDMI Port D Control
0xE1180	0xE1183	LVDS_CTL	LVDS_CTL	LVDS Port Control
0xE1200	0xE1203	VIDEO_DIP_CTL_B	VIDEO_DIP_CTL	Transcoder B Video Data Island Packet Control
0xE1208	0xE120B	VIDEO_DIP_DATA_B	VIDEO_DIP_DATA	Transcoder B Video Data Island Packet Data

Address Offset Values for SNB PCH Registers				
Start Address	**Stop Address**	**Short name**	**Format**	**Full Name**
0xE1210	0xE1213	VIDEO_DIP_GCP_B	VIDEO_DIP_GCP	Transcoder B Video Data Island Payload
0xE1300	0xE1303	TRANS_DP_CTL_B	TRANS_DP_CTL	Transcoder B DisplayPort Control
0xE2000	0xE2003	TRANS_HTOTAL_C	HTOTAL	Transcoder C Horizontal Total
0xE2004	0xE2007	TRANS_HBLANK_C	HBLANK	Transcoder C Horizontal Blank
0xE2008	0xE200B	TRANS_HSYNC_C	HSYNC	Transcoder C Horizontal Sync
0xE200C	0xE200F	TRANS_VTOTAL_C	VTOTAL	Transcoder C Vertical Total
0xE2010	0xE2013	TRANS_VBLANK_C	VBLANK	Transcoder C Vertical Blank
0xE2014	0xE2017	TRANS_VSYNC_C	VSYNC	Transcoder C Vertical Sync
0xE2028	0xE202B	TRANS_VSYNCSHIFT_C	VSYNCSHIFT	Transcoder C Vertical Sync Shift
0xE2030	0xE2033	TRANS_DATAM1_C	DATAM	Transcoder C Data M value 1
0xE2034	0xE2037	TRANS_DATAN1_C	DATAN	Transcoder C Data N value 1
0xE2038	0xE203B	TRANS_DATAM2_C	DATAM	Transcoder C Data M value 2
0xE203C	0xE203F	TRANS_DATAN2_C	DATAN	Transcoder C Data N value 2
0xE2040	0xE2043	TRANS_LINKM1_C	LINKM	Transcoder C Link M value 1
0xE2044	0xE2047	TRANS_LINKN1_C	LINKN	Transcoder C Link N value 1
0xE2048	0xE204B	TRANS_LINKM2_C	LINKM	Transcoder C Link M value 2
0xE204C	0xE204F	TRANS_LINKN2_C	LINKN	Transcoder C Link N value 2
0xE2200	0xE2203	VIDEO_DIP_CTL_C	VIDEO_DIP_CTL	Transcoder C Video Data Island Packet Control
0xE2208	0xE220B	VIDEO_DIP_DATA_C	VIDEO_DIP_DATA	Transcoder C Video Data Island Packet Data
0xE2210	0xE2213	VIDEO_DIP_GCP_C	VIDEO_DIP_GCP	Transcoder C Video Data Island Payload
0xE2300	0xE2303	TRANS_DP_CTL_C	TRANS_DP_CTL	Transcoder C DisplayPort Control
0xE4100	0xE4103	DP_CTL_B	DP_CTL	DisplayPort B Control
0xE4110	0xE4113	DP_AUX_CTL_B	DP_AUX_CTL	DisplayPort B AUX

Address Offset Values for SNB PCH Registers				
Start Address	**Stop Address**	**Short name**	**Format**	**Full Name**
				Channel Control
0xE4114	0xE4127	DP_AUX_DATA_[1-5]_B	DP_AUX_DATA_[1-5]	DisplayPort B AUX Channel Data
0xE4200	0xE4203	DP_CTL_C	DP_CTL	DisplayPort C Control
0xE4210	0xE4213	DP_AUX_CTL_C	DP_AUX_CTL	DisplayPort C AUX Channel Control
0xE4214	0xE4227	DP_AUX_DATA_[1-5]_C	DP_AUX_DATA_[1-5]	DisplayPort C AUX Channel Data
0xE4300	0xE4303	DP_CTL_D	DP_CTL	DisplayPort D Control
0xE4310	0xE4313	DP_AUX_CTL_D	DP_AUX_CTL	DisplayPort D AUX Channel Control
0xE4314	0xE4327	DP_AUX_DATA_[1-5]_D	DP_AUX_DATA_[1-5]	DisplayPort D AUX Channel Data
0xE4F00	0xE4F27	DP_BUFTRANS_[0-9]	DP_BUFTRANS_[0-9]	DP Buffer Translation
0xE5000	0xE5003	AUD_CONFIG_A	AUD_CONFIG	Audio Configuration Transcoder A
0xE5010	0xE5013	AUD_MISC_CTRL_A	AUD_MISC_CTRL	Audio Misc Control Transcoder A
0xE5020	0xE5023	AUD_VID_DID	AUD_VID_DID	Audio Vendor ID / Device ID
0xE5024	0xE5027	AUD_RID	AUD_RID	Audio Revision ID
0xE5028	0xE502B	AUD_CTS_ENABLE_A	AUD_CTS_ENABLE	Audio CTS Programming Enable Transcoder A
0xE504C	0xE504F	AUD_PWRST	AUD_PWRST	Audio Power State
0xE5050	0xE5053	AUD_HDMIW_HDMIEDID_A	AUD_HDMIW_HDMIEDID	Audio HDMI Data EDID Block Transcoder A
0xE5054	0xE5057	AUD_HDMIW_INFOFR_A	AUD_HDMIW_INFOFR	Audio Widget Data Island Packet Transcoder A
0xE507C	0xE507F	AUD_PORT_EN_HD_CFG	AUD_PORT_EN_HD_CFG	Audio Port Enable HDAudio Config
0xE5080	0xE5083	AUD_OUT_DIG_CNVT_A	AUD_OUT_DIG_CNVT	Audio Digital Converter A
0xE5084	0xE5087	AUD_OUT_STR_DESC_A	AUD_OUT_STR_DESC	Audio Stream Descriptor Format Converter A
0xE5088	0xE508B	AUD_OUT_CH_STR	AUD_OUT_CH_STR	Audio Channel ID and Stream ID
0xE50A8	0xE50AB	AUD_PINW_CONNLNG_LIST	AUD_PINW_CONNLNG_LIST	Audio Connection List
0xE50AC	0xE50AF	AUD_PINW_CONNLNG_SEL	AUD_PINW_CONNLNG_SEL	Audio Connection

Address Offset Values for SNB PCH Registers				
Start Address	**Stop Address**	**Short name**	**Format**	**Full Name**
				Select
0xE50B4	0xE50B7	AUD_CNTL_ST_A	AUD_CNTL_ST	Audio Control State Transcoder A
0xE50C0	0xE50C3	AUD_CNTRL_ST2	AUD_CNTRL_ST2	Audio Control State 2
0xE50C4	0xE50C7	AUD_CNTRL_ST3	AUD_CNTRL_ST3	Audio Control State 3
0xE50D4	0xE50D7	AUD_HDMIW_STATUS	AUD_HDMIW_STATUS	Audio HDMI Status
0xE5100	0xE5103	AUD_CONFIG_B	AUD_CONFIG	Audio Configuration Transcoder B
0xE5110	0xE5113	AUD_MISC_CTRL_B	AUD_MISC_CTRL	Audio Misc Control Transcoder B
0xE5128	0xE512B	AUD_CTS_ENABLE_B	AUD_CTS_ENABLE	Audio CTS Programming Enable Transcoder B
0xE5150	0xE5153	AUD_HDMIW_HDMIEDID_B	AUD_HDMIW_HDMIEDID	Audio HDMI Data EDID Block Transcoder B
0xE5154	0xE5157	AUD_HDMIW_INFOFR_B	AUD_HDMIW_INFOFR	Audio Widget Data Island Packet Transcoder B
0xE5180	0xE5183	AUD_OUT_DIG_CNVT_B	AUD_OUT_DIG_CNVT	Audio Digital Converter B
0xE5184	0xE5187	AUD_OUT_STR_DESC_B	AUD_OUT_STR_DESC	Audio Stream Descriptor Format Converter B
0xE51B4	0xE51B7	AUD_CNTL_ST_B	AUD_CNTL_ST	Audio Control State Transcoder B
0xE5200	0xE5203	AUD_CONFIG_C	AUD_CONFIG	Audio Configuration Transcoder B
0xE5210	0xE5213	AUD_MISC_CTRL_C	AUD_MISC_CTRL	Audio Misc Control Transcoder B
0xE5228	0xE522B	AUD_CTS_ENABLE_C	AUD_CTS_ENABLE	Audio CTS Programming Enable Transcoder B
0xE5250	0xE5253	AUD_HDMIW_HDMIEDID_C	AUD_HDMIW_HDMIEDID	Audio HDMI Data EDID Block Transcoder B
0xE5254	0xE5257	AUD_HDMIW_INFOFR_C	AUD_HDMIW_INFOFR	Audio Widget Data Island Packet Transcoder B
0xE5280	0xE5283	AUD_OUT_DIG_CNVT_C	AUD_OUT_DIG_CNVT	Audio Digital Converter B
0xE5284	0xE5287	AUD_OUT_STR_DESC_C	AUD_OUT_STR_DESC	Audio Stream Descriptor Format

Address Offset Values for SNB PCH Registers				
Start Address	**Stop Address**	**Short name**	**Format**	**Full Name**
				Converter B
0xE52B4	0xE52B7	AUD_CNTL_ST_C	AUD_CNTL_ST	Audio Control State Transcoder B
0xF0008	0xF000B	TRANS_CONF_A	TRANS_CONF	Transcoder A Config
0xF000C	0xF000F	FDI_RX_CTL_A	FDI_RX_CTL	FDI A RX Control
0xF0010	0xF0013	FDI_RX_MISC_A	FDI_RX_MISC	FDI A RX Miscellaneous
0xF0014	0xF0017	FDI_RX_IIR_A	FDI_RX_IIR	FDI A RX Interrupt Identity
0xF0018	0xF001B	FDI_RX_IMR_A	FDI_RX_IMR	FDI A RX Interrupt Mask
0xF0030	0xF0033	FDI_RX_TUSIZE_1_A	FDI_RX_TUSIZE	FDI A RX TU Size 1
0xF0038	0xF003B	FDI_RX_TUSIZE_2_A	FDI_RX_TUSIZE	FDI A RX TU Size 2
0xF1008	0xF100B	TRANS_CONF_B	TRANS_CONF	Transcoder B Config
0xF100C	0xF100F	FDI_RX_CTL_B	FDI_RX_CTL	FDI B RX Control
0xF1010	0xF1013	FDI_RX_MISC_B	FDI_RX_MISC	FDI B RX Miscellaneous
0xF1014	0xF1017	FDI_RX_IIR_B	FDI_RX_IIR	FDI B RX Interrupt Identity
0xF1018	0xF101B	FDI_RX_IMR_B	FDI_RX_IMR	FDI B RX Interrupt Mask
0xF1030	0xF1033	FDI_RX_TUSIZE_1_B	FDI_RX_TUSIZE	FDI B RX TU Size 1
0xF1038	0xF103B	FDI_RX_TUSIZE_2_B	FDI_RX_TUSIZE	FDI B RX TU Size 2
0xF2008	0xF200B	TRANS_CONF_C	TRANS_CONF	Transcoder C Config
0xF200C	0xF200F	FDI_RX_CTL_C	FDI_RX_CTL	FDI C RX Control
0xF2010	0xF2013	FDI_RX_MISC_C	FDI_RX_MISC	FDI C RX Miscellaneous
0xF2014	0xF2017	FDI_RX_IIR_C	FDI_RX_IIR	FDI C RX Interrupt Identity
0xF2018	0xF201B	FDI_RX_IMR_C	FDI_RX_IMR	FDI C RX Interrupt Mask
0xF2030	0xF2033	FDI_RX_TUSIZE_1_C	FDI_RX_TUSIZE	FDI C RX TU Size 1
0xF2038	0xF203B	FDI_RX_TUSIZE_2_C	FDI_RX_TUSIZE	FDI C RX TU Size 2

Intel® OpenSource HD Graphics Programmer's Reference Manual (PRM)

Volume 4 Part 1: Subsystem and Cores – Shared Functions (SandyBridge)

For the 2011 Intel Core Processor Family

May 2011

Revision 1.0

NOTICE:

This document contains information on products in the design phase of development, and Intel reserves the right to add or remove product features at any time, with or without changes to this open source documentation.

Doc Ref #: IHD-OS-V4 Pt1 – 05 11

Creative Commons License

You are free to Share — to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

No Derivative Works. You may not alter, transform, or build upon this work.

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

Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The SandyBridge chipset family, Havendale/Auburndale chipset family, Intel® 965 Express Chipset Family, Intel® G35 Express Chipset, and Intel® 965GMx Chipset Mobile Family Graphics Controller may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel® sales office or your distributor to obtain the latest specifications and before placing your product order. I2C is a two-wire communications bus/protocol developed by Philips. SMBus is a subset of the I2C bus/protocol and was developed by Intel®. Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

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

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

Copyright © 2011, Intel Corporation. All rights reserved.

Contents

1. Subsystem Overview 5
1.1 Introduction 5
1.2 Subsystem Topology 5
1.3 Execution Units (EUs) 5
1.4 Thread Dispatching 6
1.5 Shared Functions 6
1.6 Messages 8
1.6.1 Message Register File (MRF) 9
1.6.2 Send Instruction 9
1.6.3 Creating and Sending a Message 10
1.6.4 Message Payload Containing a Header 11
1.6.5 Writebacks 11
1.6.6 Message Delivery Ordering Rules 11
1.6.7 Execution Mask and Messages 12
1.6.8 End-Of-Thread (EOT) Message 12
1.6.9 Performance 13
1.6.10 Message Description Syntax 13
1.6.11 Message Errors 14
2. Sampling Engine 16
2.1 Texture Coordinate Processing 17
2.1.1 Texture Coordinate Normalization 17
2.1.2 Texture Coordinate Computation 17
2.2 Texel Address Generation 18
2.2.1 Level of Detail Computation (Mipmapping) 19
2.2.2 Intra-Level Filtering Setup 22
2.2.3 Texture Address Control 25
2.3 Texel Fetch 28
2.3.1 Texel Chroma Keying 29
2.4 Shadow Prefilter Compare 29
2.5 Texel Filtering 30
2.6 Texel Color Gamma Linearization 30
2.7 Multisampled Surface Behavior [DevSNB+] 31
2.8 Denoise/Deinterlacer [DevSNB] 31
2.8.1 Introduction 31
2.8.2 Denoise Algorithm 34
2.8.3 Block Noise Estimate (part of Global Noise Estimate) 38
2.8.4 Deinterlacer Algorithm 39
2.8.5 Field Motion Detector 53
2.8.6 Implementation Overview 55
2.9 Adaptive Video Scaler 57
2.9.1 Filtering Operations 59
2.10 Image Enhancement Filter and Video Signal Analysis 61
2.10.1 Block Diagram 62
2.10.2 Detail Filter Algorithm 62
2.10.3 Combination mode 64
2.11 State 69
2.11.1 BINDING_TABLE_STATE 69
2.11.2 SURFACE_STATE 70
2.11.3 SAMPLER_STATE 101

2.11.4 SAMPLER_8x8_STATE [DevSNB+] 123
2.11.5 3DSTATE_CHROMA_KEY 128
2.11.6 3DSTATE_SAMPLER_PALETTE_LOAD0 130
2.11.7 3DSTATE_SAMPLER_PALETTE_LOAD1 [DevSNB] 131
2.11.8 3DSTATE_MONOFILTER_SIZE [DevILK+] 132
2.12 Messages 133
2.12.1 Initiating Messages 133
2.12.2 Writeback Message 151
3. Data Port 166
3.1 Cache Agents 166
3.1.1 Render Cache 167
3.1.2 Sampler Cache 167
3.1.3 Constant Cache [DevSNB+] 167
3.2 Surfaces 167
3.2.1 Surface State Model 167
3.2.2 Stateless Model 168
3.3 Write Commit 168
3.4 Read/Write Ordering 169
3.5 Accessing Buffers 169
3.6 Accessing Media Surfaces 170
3.6.1 Color Processing [DevSNB+] 170
3.7 Accessing Render Targets 193
3.7.1 Single Source 193
3.7.2 Dual Source [DevSNB+] 193
3.7.3 Replicate Data 193
3.7.4 Multiple Render Targets (MRT) 194
3.8 State 194
3.8.1 BINDING_TABLE_STATE 194
3.8.2 SURFACE_STATE 194
3.8.3 COLOR_PROCESSING_STATE [DevSNB+] 194
3.9 Messages 207
3.9.1 Global Definitions 207
3.9.2 Data Port Messages 207
3.9.3 OWord Block Read/Write 213
3.9.4 Unaligned OWord Block Read [DevSNB+] 216
3.9.5 OWord Dual Block Read/Write 218
3.9.6 Media Block Read/Write 220
3.9.7 DWord Scattered Read/Write 228
3.9.8 DWord Atomic write message [DevSNB] 232
3.9.9 Render Target Write 235
3.9.10 Render Target UNORM Read/Write [DevCTG] to [DevSNB] 253
3.9.11 Streamed Vertex Buffer Write [DevSNB] 259
3.9.12 AVC Loop Filter Read [DevCTG] to [DevSNB] 260

1. Subsystem Overview

1.1 Introduction

The DevSNB (SandyBridge) subsystem consists of an array of *execution units* (*EUs*, sometimes referred to as an arrray of *cores*) along with a set of *shared functions* outside the EUs that the EUs leverage for I/O and for complex computations. Programmers access the DevSNB Subsystem via the 3D or Media pipelines.

EUs are general-purpose programmable cores that support a rich instruction set that has been optimized to support various 3D API shader languages as well as media functions (primarily video) processing.

Shared functions are hardware units which serve to provide specialized supplemental functionality for the EUs. A shared function is implemented where the demand for a given specialized function is insufficient to justify the costs on a per-EU basis. Instead a single instantiation of that specialized function is implemented as a stand-alone entity outside the EUs and shared amongst the EUs.

Invocation of the shared functionality is performed via a communication mechanism call a “message”. A message is a small, self-contained packet of information created by a kernel and directed to specific shared function. The message is defined by sequential series of MRF registers which hold message operands, a destination shared function ID, a function-specific encoding of the desired operation to be performed, and a destination GRF register to which any writeback response is to be directed. Messages are dispatched to the shared function under software control via the ‘send’ instruction. This instruction identifies the contents of the message and the GRF register location(s) to direct any response.

The message construction and delivery mechanisms are general in their definition and capable of supporting a wide variety of shared functions.

1.2 Subsystem Topology

The subsystem is organized as an array of EUs, and a set of functions that are shared among all of the EUs. (The EU array is further divided into rows with each row having its own first level instruction cache and Extended Math shared function, though this aspect of the implemented topology is not exposed to software). The Sampler, DataPort, URB and Message Gateway functions are shared among the entire array of EUs.

1.3 Execution Units (EUs)

Each EU is a vector machine capable of performing a given operation on as many as 16 pieces of data of the same type in parallel (though not necessarily on the same instant in time). In addition, each EU can support a number of execution contexts called *threads* that are used to avoid stalling the EU during a high-latency operation (external to the EU) by providing an opportunity for the EU to switch to a completely different workload with minimal latency while waiting for the high-latency operation to complete.

For example, if a program executing on an EU requires a texture read by the sampling engine, the EU may not necessarily idle while the data is fetched from memory, arranged, filtered and returned to the EU. Instead the EU will likely switch execution to another (unrelated) thread associated with that EU. If that thread encounters a stall, the EU may switch to yet another thread and so on. Once the Sampler result arrives back at the EU, the EU can switch back to the original thread and use the returned data as it continues execution of that thread.

The fact that there are multiple EU cores each with multiple threads can generally be ignored by software. There are some exceptions to this rule: e.g., for

- thread-to-thread communication (see *Message Gateway*, *Media*)
- synchronization of thread output to memory buffers (see *Geometry Shader*).

In contrast, the internal SIMD aspects of the EU are very much exposed to software.

This volume will not deal with the details of the EUs.

1.4 Thread Dispatching

When the 3D and Media pipelines send requests for thread initiation to the Subsystem, the thread Dispatcher receives the requests. The dispatcher performs such tasks as arbitrating between concurrent requests, assigning requested threads to hardware threads on EUs, allocating register space in each EU among multiple threads, and initializing a thread's registers with data from the fixed functions and from the URB. This operation is largely transparent to software.

1.5 Shared Functions

In general, a shared function has the ability to receive messages at its input, perform some specialized amount of work for each, and if required, generate output back to the message's originating execution unit (Message Gateway may generate output to a target execution unit specified by the message).

To uniquely identify shared functions, each is assigned a unique 4-bit identifier code called its 'Function ID'. This ID is specified in the 'send' instruction's 32b <desc> field of each message. DevSNB Function ID assignments are listed in the *Graphics Processing Engine* chapter of this specification.

Each shared function may support one or more related operations within itself. For example an Extended Math shared function may support operations such as reciprocal, sine, cosine, and/or others. These are generically referred to as sub-functions. The communication method as to which sub-function is desired is typically contained in the 16b 'function-control' field of the 'send' instruction <desc> field. Alternatively, a function may choose to define sub-function encodings in-band within message payload, or in the case of a single function shared-function, the function code may be implied. The architecture in no way interprets the sub-function code and the actual implementation choice is left to the function itself.

The Shared Function units included in the Subsystem are as follows (refer to the chapters devoted to each of these functions):

- Extended Math function
- Sampling Engine function

- DataPort function
- Message Gateway function
- Unified Return Buffer (URB)
- Thread Spawner (TS)
- Null function

The **Extended Math** function acts as an extension of the math functions already available inside the EUs. Certain functions such as inverse, square root, exponentiation, etc., require significant hardware resources to implement and are used infrequently enough that it is inefficient to implement them separately in each EU. The EUs therefore send the operands for these operations along with the operation to be performed to the Extended Math function which computes and returns the result to the requesting EU.

The **Sampling Engine** acts a (read-only) I/O port on behalf of the EUs, translating texture coordinates (and/or structure references) to memory addresses, reading texels and/or other data from memory, and in the case of texels, combining and filtering them according to programmed state. The resulting pixel and/or other data are then returned to the requesting EU.

The **Data Port** function acts as another I/O port on behalf of the EUs. It is both a read and a write port, and the only way for the Graphics Processing Engine to write results (e.g., images) back to memory. The Data Port contains the render and depth caches which receive the newly rendered pixels and write them out to memory when necessary. They also permit previously rendered objects to be read back efficiently by the Graphics Processing Engine in order to blend them with other rendered objects and test for visibility of newly rendered objects. Finally, the Data Port also provides read access constant buffers (arrays of constants in memory.)

The **Message Gateway** allows a thread to communicate (send a message to) another thread. A key is used to connect the sender and receiver threads, and a simple gateway protocol is used to send messages. This is primarily intended for media where a parent/child thread model is sometimes used and requires parent and child threads to synchronize and efficiently share information. It is not intended to be used by 3D graphics rendering threads.

The **Unified Return Buffer** (URB) is a single set of registers that EU threads use to return result data for future fixed functions and their threads to make use of. Individual entries in the buffer are “owned” by a given fixed function but a mechanism is provided where other fixed functions (those that follow) can read the data placed there by another fixed function. The buffer is considered a “Shared Function” since EUs need to be able to write result data to it using messages. In general, EU threads write their final results either to memory via the Data Port or to the URB for re-use by subsequent EU threads or certain 3D pipeline fixed-function units (CLIP, GS).

The **Thread Spawner** (TS) is a Shared Function that acts as a conduit for dispatching kernel-software-generated threads, one thread can request another thread to be dispatched by sending a request to the TS. TS is unique as it is also a Fixed Function in the media pipeline for dispatching threads originated from Video Front End fixed function.

The **Null** shared function is supported to allow the broadcast of certain information (e.g, End Of Thread) without invoking any other operation or response.

1.6 Messages

Communication between the EUs and the shared functions and between the fixed function pipelines (which are not considered part of the "Subsystem") and the EUs is accomplished via packets of information called *messages*. Message transmission is requested via the 'send' instruction. Refer to the 'send' instruction definition in the *ISA Reference* chapter for details.

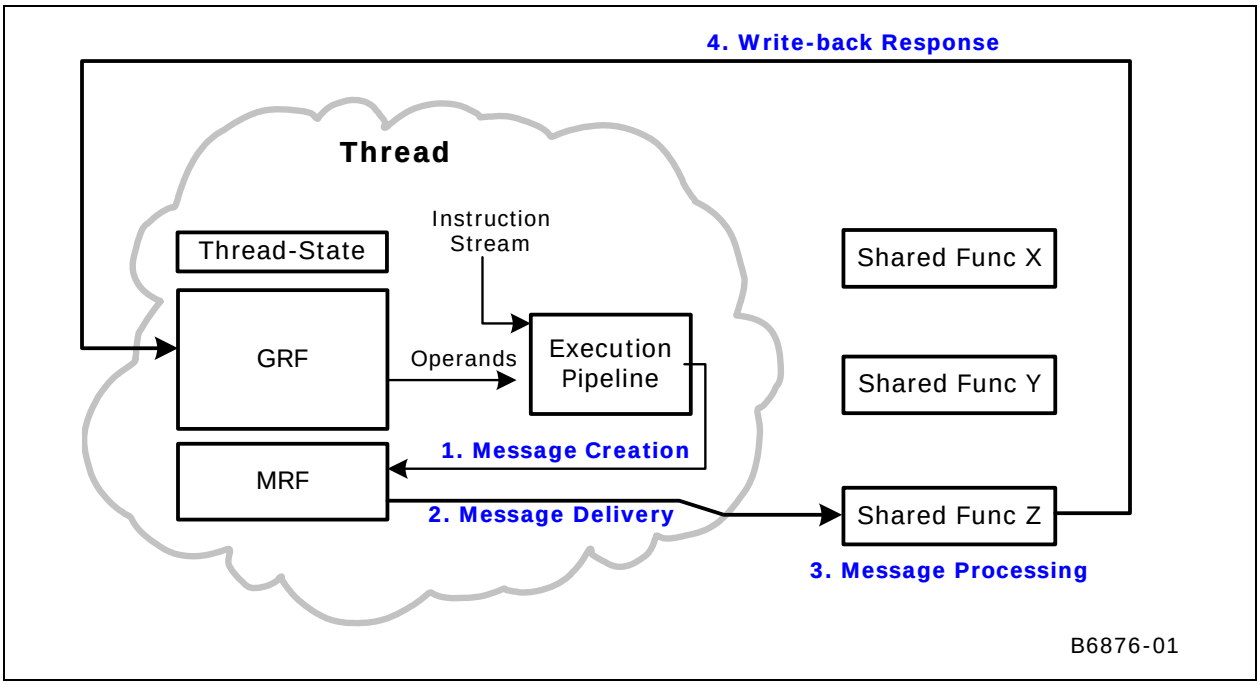
The information transmitted in a message falls into two categories:

- **Message Payload** data sourced from some number of registers (from 1 to 15 registers) in the Message Register File (MRF). The contents of the payload are dependent on the target function and specific function (etal), and may contain a header portion and/or data portion.
- Associated ("sideband") information provided by:
    - **Message Descriptor** specified with the 'send' instruction. Included in the message descriptor is control and routing information such as the target function ID, message payload length, response length, etc.
    - Additional information provided by the 'send' instruction, e.g., the starting destination register number, the execution mask (EMASK), etc.
    - A small subset of Thread State, such as the Thread ID, EUID, etc.

The software view of messages is shown in Figure 1-1. There are four basic phases to a message's lifetime as illustrated below:

1. Creation — The thread assembles the message payload into the Message Register File (MRF). This is done by a series of one or more instruction which specify a MRF register as the destination.
2. Delivery — The thread issues the message for delivery via the 'send' instruction. The 'send' instruction specifies the MRF register which is the first of a sequential register series which makes the data payload, the length of the message payload within the MRF, the destination shared function ID (SFID), and where in the GRF any response is to be directed. The messaging subsystem will enqueue the message for delivery and eventually route the message to the specified shared function.
3. Processing — The shared function receives the message and services it accordingly, as defined by the shared function definition.
4. Writeback — If called for, the shared function delivers an integral number of registers of data to the thread's GRF in response to the message.

Figure 1-1. Data Flow Associated With Messages



1.6.1 Message Register File (MRF)

Each thread has a dedicated MRF which is logically identical to the GRF: 256 bits wide per register, with word-wide addressability. There are 16 MRF registers, referred to as “m0”..”m15”. From a software perspective, the MRF is write-only and thus may only be used as a destination specifier. Limited register-region specifications are allowed so long as the region is contained within a single MRF register.

Each register of the MRF has an associated in-flight status, indicating the contents of the register is needed as part of a pending message, but has yet to be transmitted by the hardware. This bit is set at the time the message is enqueued for delivery via the ‘send’ instruction. Should a subsequent write to an in-flight register be attempted, the execution unit will temporarily suspend the thread’s execution until the register’s in-flight status is cleared (i.e., the message has been transmitted).

Normal threads should construct their messages in m1..m15. The thread is free to start a message payload at any MRF register location, even to the point of having multiple messages under construction at the same time in non-overlapping spaces in the MRF. Further multiple messages over non-overlapping MRF space can be enqueued awaiting transmission at the same time. Regardless of actual hardware implementation, the thread should not assume that MRF addresses above m15 wrap to legal MRF registers.

1.6.2 Send Instruction

Messages are sent programmatically by the thread through the ‘send’ instruction. This instruction enqueues a message for delivery and marks as in-flight all MRF registers used for the message payload. It also allows for an optional implied move of one GRF register to a MRF register prior to the message being issued. This implied move allows for a higher message performance, eliminating the explicit ‘mov’

that would normally be required to move R0 to the lead MRF register of the message (as required by many message definitions).

A typical 'send' instruction is exemplified here (please see the ISA for a full instruction description). This example performs an implicit move from r0 to m3, then issues a message to the Extended Math unit, with a payload of 1 register starting at m3, and expecting 1 register in reply to be placed in r5.

```
send (16) r5 m3 r0 0x01110001
```

The execution unit guarantees that any prior instruction which wrote to a MRF register is guaranteed to have retired, and its result written to the destination MRF register in time for message transmission.

1.6.3 Creating and Sending a Message

A code snippet is listed below, showing a 4-register message (m3 to m6) whose response is directed to r30. Note that message construction does not have to occur in MRF register order.

```
...
mul (8)   m4    r20     r19
mov (8)   m6    r21
add (8)   m5    r29     r28
send (8)  r30   m3      r0   <desc>
...
```

Once a 'send' instruction is issued, the MRF registers used for its payload are marked as 'in-flight'. These registers remain in this state until the message is actually transmitted to the shared function and the register contents are no longer need. Any subsequent write to a MRF register which is in-flight results in a dependency and a thread switch until such time that the in-flight condition is cleared. An example is shown below in which the attempt to re-use m6 may result in a thread switch until message 1 is transmitted.

```
...
// --- message 1 ---
mul (8)   m4    r20     r19
mov (8)   m6    r21
add (8)   m5    r29     r28
send (8)  r30   m3      r0   <desc>
...

// --- message 2 ---
mov (8)   m6    r15     // thread switch until the
                        // previous msg is sent and
                        // m6 in-flight is cleared.
...
```

MRF registers of one message may be reused for a subsequent message without restriction. The in-flight check mechanism prevents a MRF register staged as part of a pending message from being altered while awaiting transmission. Further, a thread may rely on the contents of a MRF register being unaltered after message transmission. This allows the thread to quickly issue an identical or slightly altered message using the same MRF register set without having to re-construct the entire payload.

Although more than one message may be enqueued at any point in time, care must be taken by the programmer to ensure that each message's destination GRF register region, if any, does no over lap with that of another enqueued message. This condition is not checked by HW. Due to varying latencies between two messages, and out-of-order, non-contiguous writeback cycles in the current implementation, the outcome in the GRF is indeterminate; It may be the result from the first message, or the result from the second message, or a mixture of data from both.

1.6.4 Message Payload Containing a Header

For most shared functions, the first register of the message payload contains the *header payload* of the message (or simply the *message header*). It contains the state fields (such as binding table pointer, sampler state pointer, etc) following a consistent format structure. Consequently, the rest of the message payload is referred to as the *data payload*.

Messages to Extended Math do not have a header and only contain data payload. Those messages may be referred to as header-less messages. Messages to Gateway combine the header and data payloads in a single message register.

1.6.5 Writebacks

Some messages generate return data as dictated by the 'function-control' (opcode) field of the 'send' instruction (part of the <desc> field). The DevSNB execution unit and message passing infrastructure do not interpret this field in any way to determine if writeback data is to be expected. Instead explicit fields in the 'send' instruction to the execution unit the starting GRF register and count of returning data. The execution unit uses this information to set in-flight bits on those registers to prevent execution of any instruction which uses them as an operand until the register(s) is(are) eventually written in response to the message. If a message is not expected to return data, the 'send' instruction's writeback destination specifier (<post_dest>) must be set to 'null' and the response length field of <desc> must be 0 (see 'send' instruction for more details).

The writeback data, if called for, arrives as a series of register writes to the GRF at the location specified by the starting GRF register and length as specified in the 'send' instruction. As each register is written back to the GRF, its in-flight flag is cleared and it becomes available for use as an instruction operand. If a thread was suspended pending return of that register, the dependency is lifted and the thread is allowed to continue execution (assuming no other dependency for that thread remains outstanding).

1.6.6 Message Delivery Ordering Rules

All messages between a thread and an individual shared function are delivered in the ordered they were sent. Messages to different shared functions originating from a single thread may arrive at their respective shared functions out of order.

The writebacks of various messages from the shared functions may return in any order. Further individual destination registers resulting from a single message may return out of order, potentially allowing

execution to continue before the entire response has returned (depending on the dependency chain inherent in the thread).

1.6.7 Execution Mask and Messages

The DevSNB Architecture defines an Execution Mask (EMask) for each instruction issued. This 16b bit-field identifies which SIMD computation channels are enabled for that instruction. Since the 'send' instruction is inherently scalar, the EMask is ignored as far as instruction dispatch is concerned. Further the execution size has no impact on the size of the 'send' instruction's implicit move (it is always 1 register regardless of specified execution size).

The 16b EMask is forwarded with the message to the destination shared function to indicate which SIMD channels were enabled at the time of the 'send'. A shared function may interpret or ignore this field as dictated by the functionality it exposes. For instance, the Extended Math shared function observes this field and performs the specified operation only on the operands with enabled channels, while the DataPort writes to the render cache ignore this field completely, instead using the pixel mask included in-band in the message payload to indicate which channels carry valid data.

1.6.8 End-Of-Thread (EOT) Message

The final instruction of all threads must be a 'send' instruction which signals 'End-Of-Thread' (EOT). An EOT message is one in which the EOT bit is set in the 'send' instruction's 32b <desc> field. When issuing instructions, the EU looks for an EOT message, and when issued, shuts down the thread from further execution and considers the thread completed.

Only a subset of the shared functions can be specified as the target function of an EOT message, as shown in the table below.

Target Shared Functions supporting EOT messages	Target Shared Functions not supporting EOT messages
Null, DataPortWrite, URB, MessageGateway, ThreadSpawner	DataPortRead, Sampler

Both the fixed-functions and the thread dispatcher require EOT notification at the completion of each thread. The thread dispatcher and fixed functions in the 3D pipeline obtain EOT notification by snooping all message transmissions, regardless of the explicit destination, looking for messages which signal end-of-thread. The Thread Spawner in the media pipeline does not snoop for EOT. As it is also a shared function, all threads generated by Thread Spawner must send a message to Thread Spawner to explicity signal end-of-thread.

The thread dispatcher, upon detecting an end-of-thread message, updates its accounting of resource usage by that thread, and is free to issue a new thread to take the place of the ended thread. Fixed functions require end-of-thread notification to maintain accounting as to which threads it issued have completed and which remain outstanding, and their associated resources such as URB handles.

Unlike the thread dispatcher, fixed-functions discriminate end-of-thread messages, only acting upon those from threads which they originated, as indicated by the 4b fixed-function ID present in R0 of end-of-thread

message payload. This 4b field is attached to the thread at new-thread dispatch time and is placed in its designated field in the R0 contents delivered to the GRF. Thus to satisfy the inclusion of the fixed-function ID, the typical end-of-thread message generally supplies R0 from the GRF as the first register of an end-of-thread message.

As an optimization, an end-of-thread message may be overload upon another "productive" message, saving the cost in execution and bandwidth of a dedicated end-of-thread message. Outside of the end-of-thread message, most threads issue a message just prior to their termination (for instance, a Dataport write to the framebuffer) so the overloaded end-of-thread is the common case. The requirement is that the message contains R0 from the GRF (to supply the fixed-function ID), and that destination shared function be either (a) the URB; (b) the Read or Write Dataport; or, (c) the Gateway, as these functions reside on the O-Bus. In the case where the last real message of a thread is to some other shared function, the thread must issue a separate message for the purposes of signaling end-of-thread to the "null" shared function.

1.6.9 Performance

The DevSNB Architecture imposes no requirement as to a shared function's latency or throughput. Due to this as well as factors such as message queuing, shared bus arbitration, implementation choices in bus bandwidth, and instantaneous demand for that function, the latency in delivering and obtaining a response to a message is non-deterministic. It is expected that a DevSNB implementation has some notion of fairness in transmission and servicing of messages so as to keep latency outliers to a minimum.

Other factors to consider with regard to performance:

- A thread may choose to have multiple messages under construction in non-overlapping registers the MRF at the same time.
- Multiple messages are allowed to be enqueued for transmission at the same time, so long as their MRF payload registers do not overlap.
- Messages may rely on the MRF registers being maintained across a send message, thus constructing subsequent messages overlaid on portions of a previous message,
- Software prefetching techniques may be beneficial for long latency data fetches (i.e. issue a load early in the thread for data that is required late in the thread).

1.6.10 Message Description Syntax

All message formats are defined in terms of DWords (32 bits). The message registers in all cases are 256 bits wide, or 8 DWords. The registers and DWords within the registers are named as follows, where n is the register number, and d is the DWord number from 0 to 7, from the least significant DWord at bits [31:0] within the 256-bit register to the most significant DWord at bits [255:224], respectively. For writeback messages, the register number indicates the offset from the specified starting destination register.

Dispatch Messages: Rn.d

Dispatch messages are sent by the fixed functions to dispatch threads. See the fixed function chapters in the *3D and Media* volume.

SEND Instruction Messages: Mn.d

These are the messages initiated by the thread via the SEND instruction to access shared functions. See the chapters on the shared functions later in this volume.

Writeback Messages: Wn.d

These messages return data from the shared function to the GRF where it can be accessed by thread that initiated the message.

The bits within each DWord are given in the second column in each table.

1.6.11 Message Errors

Messages are constructed via software, and not all possible bit encodings are legal, thus there is the possibility that a message may be sent containing one or more errors in its descriptor or payload contents. There are two points of error detection in the message passing system: (a) the message delivery subsystem is capable of detecting bad FunctionIDs and some cases of bad message lengths; (b) the shared functions contain various error detection mechanisms which identify bad sub-function codes, bad message lengths, and other misc errors. The error detection capabilities are specific to each shared function. The execution unit hardware itself does not perform message validation prior to transmission.

In both cases, information regarding the erroneous message is captured and made visible through MMIO registers, and the driver notified via an interrupt mechanism. The set of possible errors is listed in Table 1-1 with the associated outcome. Please see the chapter on error handling for detailed information.

Table 1-1. Error Cases

Error	Outcome
Bad Shared Function ID	The message is discarded before reaching any shared function. If the message specified a destination, those registers will be marked as in-flight, and any future usage by the thread of those registers will cause a dependency which will never clear, resulting in a hung thread and eventual time-out.
Unknown opcode Incorrect message length	The destination shared function detects unknown opcodes (as specified in the 'send' instructions <desc> field), and known opcodes where the message payload is either too long or too short, and threats these cases as errors. When detected, the shared function latches and makes available via MMIO registers the following information: the EU and thread ID which sent the message, the length of the message and expected response, and any relevant portions of the first register (R0) of the message payload. The shared function alerts the driver of an erroneous message through and interrupt mechanism, then continues normal operation with the subsequent message.
Bad message contents in payload	Detection of bad data is an implementation decision of the shared function. Not all fields may be checked by the shared function, so an erroneous payload may return bogus data or no data at all. If an erroneous value is detected by the shared function, it is free to discard the message and continue with the subsequent message. If the thread was expecting a response, the destination registers specified in the associated 'send' instruction are never cleared potentially resulting in a hung thread and time-out.

Error	Outcome
Incorrect response length	Case: too few registers specified – the thread may proceed with execution prior to all the data returning from the shared function, resulting in the thread operating on bad data in the GRF. Case: too many registers specified – the message response does not clear all the registers of the destination. In this case, if the thread references any of the residual registers, it may hand and result in an eventual time-out.
Improper use of End-Of-Thread (EOT)	Any 'send' instruction which specifies EOT must have a 'null' destination register. The EU enforces this and, if detected, will not issue the 'send' instruction, resulting in a hung thread and an eventual time-out. The 'send' instruction specifies that EOT is only recognized if the <desc> field of the instruction is an immediate. Should a thread attempt to end a thread using a <desc> sourced from a register, the EOT bit will not be recognized. In this case, the thread will continue to execute beyond the intended end of thread, resulting in a wide range of error conditions.
Two outstanding messages using overlapping GRF destinations ranges	This is not checked by HW. Due to varying latencies between two messages, and out-of-order, non-contiguous writeback cycles, the outcome in the GRF is indeterminate; may be the result from the first message, or the result from the second message, or a combination of both.

2. Sampling Engine

The Sampling Engine provides the capability of advanced sampling and filtering of surfaces in memory.

The sampling engine function is responsible for providing filtered texture values to the DevSNB Core in response to sampling engine messages.. The sampling engine uses SAMPLER_STATE to control filtering modes, address control modes, and other features of the sampling engine. A pointer to the sampler state is delivered with each message, and an index selects one of 16 states pointed to by the pointer. Some messages do not require SAMPLER_STATE. In addition, the sampling engine uses SURFACE_STATE to define the attributes of the surface being sampled. This includes the location, size, and format of the surface as well as other attributes.

Although data is commonly used for "texturing" of 3D surfaces, the data can be used for any purpose once returned to the execution core.

The following table summarizes the various subfunctions provided by the Sampling Engine. After the appropriate subfunctions are complete, the 4-component (reduced to fewer components in some cases) filtered texture value is provided to the DevSNB Core in order to complete the *sample* instruction.

Subfunction	Description
Texture Coordinate Processing	Any required operations are performed on the incoming pixel's interpolated internal texture coordinates. These operations may include: cube map intersection.
Texel Address Generation	The Sampling Engine will determine the required set of texel samples (specific texel values from specific texture maps), as defined by the texture map parameters and filtering modes. This includes coordinate wrap/clamp/mirror control, mipmap LOD computation and sample and/or miplevel weighting factors to be used in the subsequent filtering operations.
Texel Fetch	The required texel samples will be read from the texture map. This step may require decompression of texel data. The texel sample data is converted to an internal format.
Texture Palette Lookup	For streams which have "paletted" texture surface formats, this function uses the "index" values read from the texture map to look up texel color data from the texture palette.
Shadow Pre-Filter Compare	For shadow mapping, the texel samples are first compared to the 3rd (R) component of the pixel's texture coordinate. The boolean results are used in the texture filter.
Texel Filtering	Texel samples are combined using the filter weight coefficients computed in the Texture Address Generation function. This "combination" ranges from simply passing through a "nearest" sample to blending the results of anisotropic filters performed on two mipmap levels. The output of this function is a single 4-component texel value.
Texel Color Gamma Linearization	Performs optional gamma decorrection on texel RGB (not A) values.
Denoise/ Deinterlacer	Performs denoise and deinterlacing functions for video content (**[DevILK+]**)
8x8 Video Scaler	Performs scaling using an 8x8 filter (**[DevILK+]**)

Subfunction	Description
Image Enhancement Filter / Video Signal Analysis	Image Enhancement functions for video content (**[DevILK+]**)

2.1 Texture Coordinate Processing

The Texture Coordinate Processing function of the Sampling Engine performs any operations on the texture coordinates that are required before physical addresses of texel samples can be generated.

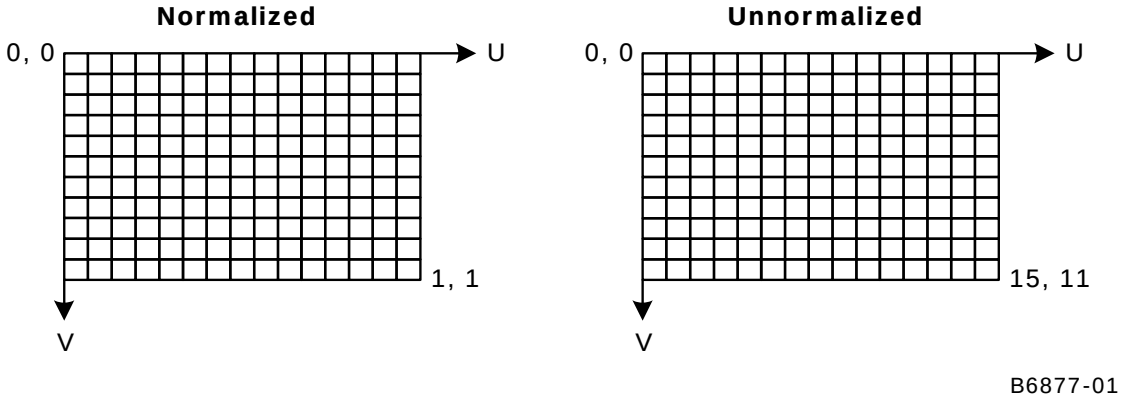
2.1.1 Texture Coordinate Normalization

A texture coordinate may have *normalized* or *unnormalized* values. In this function, unnormalized coordinates are normalized.

Normalized coordinates are specified in units relative to the map dimensions, where the origin is located at the upper/left edge of the upper left texel, and the value 1.0 coincides with the lower/right edge of the lower right texel . 3D rendering typically utilizes normalized coordinates.

Unnormalized coordinates are in units of texels and have not been divided (normalized) by the associated map's height or width. Here the origin is the located at the upper/left edge of the upper left texel of the base texture map. Unnormalized coordinates delivered to the sampling engine are only supported with the "ld" type messages.

Figure 2-1. Normalized vs. Unnormalized Texture Coordinates



2.1.2 Texture Coordinate Computation

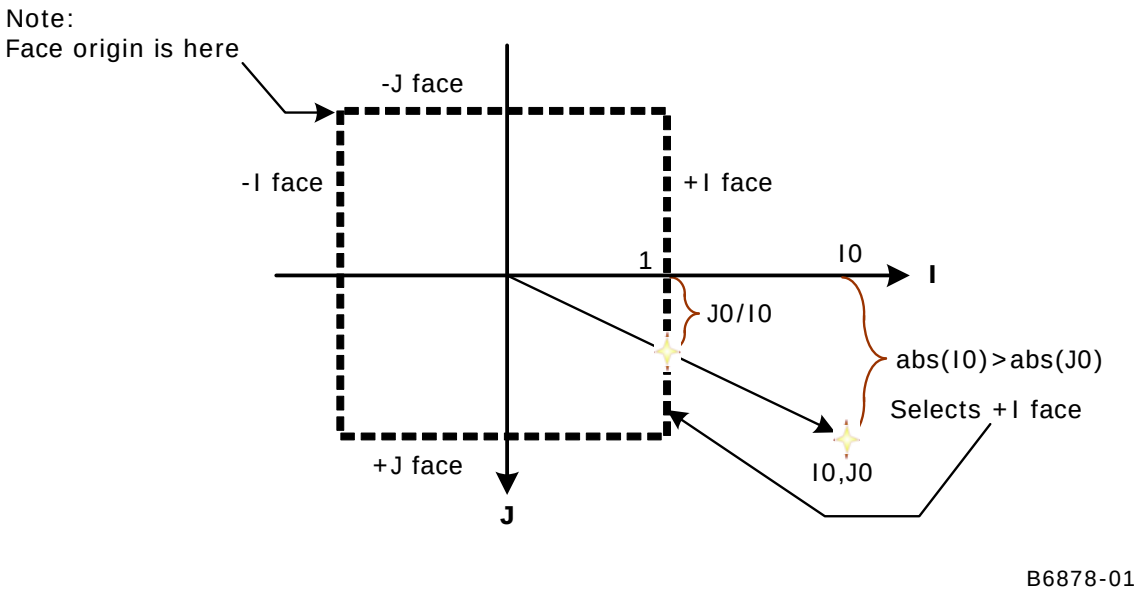
Cartesian (2D) and homogeneous (projected) texture coordinate values are projected from (interpolated) screen space back into texture coordinate space by dividing the pixel's S and T components by the Q component. This operation is done as part of the pixel shader kernel in the DevSNB Core.

Vector (cube map) texture coordinates are generated by first determining which of the 6 cube map faces (+X, +Y, +Z, -X, -Y, -Z) the vector intersects. The vector component (X, Y or Z) with the largest absolute

value determines the proper (major) axis, and then the sign of that component is used to select between the two faces associated with that axis. The coordinates along the two minor axes are then divided by the coordinate of the major axis, and scaled and translated, to obtain the 2D texture coordinate ([0,1]) within the chosen face. Note that the coordinates delivered to the sampling engine must already have been divided by the component with the largest absolute value.

An illustration of this cube map coordinate computation, simplified to only two dimensions, is provided below:

Figure 2-2. Cube Map Coordinate Computation Example



2.2 Texel Address Generation

To better understand texture mapping, consider the mapping of each object (screen-space) pixel onto the textures images. In texture space, the pixel becomes some arbitrarily sized and aligned quadrilateral. Any given pixel of the object may “cover” multiple texels of the map, or only a fraction of one texel. For each pixel, the usual goal is to sample and filter the texture image in order to best represent the covered texel values, with a minimum of blurring or aliasing artifacts. Per-texture state variables are provided to allow the user to employ quality/performance/footprint tradeoffs in selecting how the particular texture is to be sampled.

The Texel Address Generation function of the Sampling Engine is responsible for determining how the texture maps are to be sampled. Outputs of this function include the number of texel samples to be taken, along with the physical addresses of the samples and the filter weights to be applied to the samples after they are read. This information is computed given the incoming texture coordinate and gradient values, and the relevant state variables associated with the sampler and surface. This function also applies the texture coordinate address controls when converting the sample texture coordinates to map addresses.

2.2.1 Level of Detail Computation (Mipmapping)

Due to the specification and processing of texture coordinates at object vertices, and the subsequent object warping due to a perspective projection, the texture image may become *magnified* (where a texel covers more than one pixel) or *minified* (a pixel covers more than one texel) as it is mapped to an object. In the case where an object pixel is found to cover multiple texels (texture minification), merely choosing one (e.g., the texel sample nearest to the pixel's texture coordinate) will likely result in severe aliasing artifacts.

Mipmapping and texture filtering are techniques employed to minimize the effect of undersampling these textures. With mipmapping, software provides *mipmap levels,* a series of pre-filtered texture maps of decreasing resolutions that are stored in a fixed (monolithic) format in memory. When mipmaps are provided and enabled, and an object pixel is found to cover multiple texels (e.g., when a textured object is located a significant distance from the viewer), the device will sample the mipmap level(s) offering a texel/pixel ratio as close to 1.0 as possible.

The device supports up to 14 mipmap levels per map surface, ranging from 8192 x 8192 texels to a 1 X 1 texel. Each successive level has ½ the resolution of the previous level in the U and V directions (to a minimum of 1 texel in either direction) until a 1x1 texture map is reached. The dimensions of mipmap levels need not be a power of 2.

Each mipmap level is associated with a *Level of Detail (LOD)* number. LOD is computed as the approximate, $\log_2$ measure of the ratio of texels per pixel. The highest resolution map is considered LOD 0. A larger LOD number corresponds to lower resolution mip level.

The *Sampler[]BaseMipLevel* state variable specifies the LOD value at which the minification filter vs. the magnification filter should be applied.

When the texture map is magnified (a texel covers more than one pixel), the base map (LOD 0) texture map is accessed, and the magnification mode selects between the nearest neighbor texel or bilinear interpolation of the 4 neighboring texels on the base (LOD 0) mipmap.

2.2.1.1 Base Level Of Detail (LOD)

The per-pixel LOD is computed in an implementation-dependent manner and approximates the $\log_2$ of the texel/pixel ratio at the given pixel. The computation is typically based on the differential texel-space distances associated with a one-pixel differential distance along the screen x- and y-axes. These texel-space distances are computed by evaluating neighboring pixel texture coordinates, these coordinates being in units of texels on the base MIP level (multiplied by the corresponding surface size in texels). The q coordinates represent the third dimension for 3D (volume) surfaces, this coordinate is a constant 0 for 2D surfaces.

The ideal LOD computation is included below.

$$LOD(x,y) = \log_2[\rho(x,y)]$$

$$\text{where:}$$

$$\rho(x,y) = \max\left\{\sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial q}{\partial x}\right)^2}, \sqrt{\left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 + \left(\frac{\partial q}{\partial y}\right)^2}\right\},$$

2.2.1.2 LOD Bias

A biasing offset can be applied to the computed LOD and used to artificially select a higher or lower miplevel and/or affect the weighting of the selected mipmap levels. Selecting a slightly higher mipmap level will trade off image blurring with possibly increased performance (due to better texture cache reuse). Lowering the LOD tends to sharpen the image, though at the expense of more texture aliasing artifacts.

The LOD bias is defined as sum of the *LODBias* state variable and the *pixLODBias* input from the input message (which can be non-zero only for sample_b messages). The application of LOD Bias is unconditional, therefore these variables must both be set to zero in order to prevent any undesired biasing.

Note that, while the LOD Bias is applied prior to clamping and min/mag determination and therefore can be used to control the min-vs-mag crossover point, its use has the undesired effect of actually changing the LOD used in texture filtering.

2.2.1.3 LOD Pre-Clamping

The LOD Pre-Clamping function can be enabled or disabled via the *LODPreClampEnable* state variable. Enabling pre-clamping matches OpenGL semantics, while disabling it matches Direct3D.

After biasing and/or adjusting of the LOD , the computed LOD value is clamped to a range specified by the (integer and fractional bits of) *MinLOD* and *MaxLOD* state variables prior to use in Min/Mag Determination.

MaxLOD specifies the lowest resolution mip level (maximum LOD value) that can be accessed, even when lower resolution maps may be available. Note that this is the only parameter used to specify the number of valid mip levels that be can be accessed, i.e., there is no explicit “number of levels stored in memory” parameter associated with a mip-mapped texture. All mip levels from the base mip level map through the level specified by the integer bits of *MaxLOD* must be stored in memory, or operation is UNDEFINED.

MinLOD specifies the highest resolution mip level (minimum LOD value) that can be accessed, where LOD==0 corresponds to the base map. This value is primarily used to deny access to high-resolution mip levels that have been evicted from memory when memory availability is low.

MinLOD and *MaxLOD* have both integer and fractional bits. The fractional parts will limit the inter-level filter weighting of the highest or lowest (respectively) resolution map. For example if *MinLOD* is 4.5 and *MipFilter* is LINEAR, LOD 4 can contribute only up to 50% of the final texel color.

2.2.1.4 Min/Mag Determination

The biased and clamped LOD is used to determine whether the texture is being minified (scaled down) or magnified (scaled up).

The *BaseMipLevel* state variable is subtracted from the biased and clamped LOD. The *BaseMipLevel* state variable therefore has the effect of selecting the “base” mip level used to compute Min/Map Determination. (This was added to match OpenGL semantics). Setting *BaseMipLevel* to 0 has the effect of using the highest-resolution mip level as the base map.

If the biased and clamped LOD is non-positive, the texture is being magnified, and a single (high-resolution) miplevel will be sampled and filtered using the *MagFilter* state variable. At this point the computed LOD is reset to 0.0. Note that LOD Clamping can restrict access to high-resolution miplevels.

If the biased LOD is positive, the texture is being minified. In this case the *MipFilter* state variable specifies whether one or two mip levels are to be included in the texture filtering, and how that (or those) levels are to be determined as a function of the computed LOD.

2.2.1.5 LOD Computation Pseudocode

This section illustrates the LOD biasing and clamping computation in pseudocode, encompassing the steps described in the previous sections. The computation of the initial per-pixel LOD value *LOD* is not shown.

[DevSNB]

Bias: S4.6

MinLod: U4.6

MaxLod: U4.6

Base: U4.1

MIPCnt: U4

SurfMinLod: U4

ResMinLod: hard-wired to zero

If Out_of_Bounds is true, LOD is set to zero and instead of sampling the surface the texels are replaced with zero in all channels, except for surface formats that don't contain alpha, for which the alpha channel is replaced with one. These texels then proceed through the rest of the pipeline.

[DevSNB] Errata: Incorrect behavior is observed in cases where the min and mag mode filters are different and SurfMinLOD is nonzero. The determination of MagMode uses the following equation instead of the one in the above pseudocode: MagMode = (LOD + SurfMinLOD – Base <= 0)

Inter-Level Filtering Setup

The *MipFilter* state variable determines if and how texture mip maps are to be used and combined. The following table describes the various mip filter modes:

***MipFilter* Value**	**Description**
MIPFILTER_NONE	Mipmapping is DISABLED. Apply a single filter on the highest resolution map available (after LOD clamping).
MIPFILTER_NEAREST	Choose the nearest mipmap level and apply a single filter to it. Here the biased LOD will be rounded to the nearest integer to obtain the desired miplevel. LOD Clamping may further restrict this miplevel selection.
MIPFILTER_LINEAR	Apply a filter on the two closest mip levels and linear blend the results using the distance between the computed LOD and the level LODs as the blend factor. Again, LOD Clamping may further restrict the selection of miplevels (and the blend factor between them).

When minifying and MIPFILTER_NEAREST is selected, the computed LOD is rounded to the nearest mip level.

When minifying and MIPFILTER_LINEAR is selected, the fractional bits of the computed LOD are used to generate an inter-level blend factor. The LOD is then truncated. The mip level selected by the truncated LOD, and the next higher (lower resolution) mip level are determined.

Regardless of *MipFilter* and the min/mag determination, all computed LOD values (two for MIPFILTER_LINEAR, otherwise one) are then unconditionally clamped to the range specified by the (integer bits of) *MinLOD* and *MaxLOD* state variables.

2.2.2 Intra-Level Filtering Setup

Depending on whether the texture is being minified or magnified, the *MinFilter* or *MagFilter* state variable (respectively) is used to select the sampling filter to be used within a mip level (intra-level, as opposed to any inter-level filter). Note that for volume maps, this selection also applies to filtering between layers.

The processing at this stage is restricted to the selection of the filter type, computation of the number and texture map coordinates of the texture samples, and the computation of any required filter parameters. The filtering of the samples occurs later on in the Sampling Engine function.

The following table summarizes the intra-level filtering modes.

Sampler[]Min/MagFilter value	Description
MAPFILTER_NEAREST	Supported on all surface types. The texel nearest to the pixel's U,V,Q coordinate is read and output from the filter.
MAPFILTER_LINEAR	Not supported on buffer surfaces. The 2, 4, or 8 texels (depending on 1D, 2D/CUBE, or 3D surface, respectively) surrounding the pixel's U,V,Q coordinate are read and a linear filter is applied to produce a single filtered texel value.
MAPFILTER_ANISOTROPIC	Not supported on buffer or 3D surfaces. A projection of the pixel onto the texture map is generated and "subpixel" samples are taken along the major axis of the projection (center axis of the longer dimension). The outermost subpixels are weighted according to closeness to the edge of the projection, inner subpixels are weighted equally. Each subpixel samples a bilinear 2x2 of texels and the results are blended according to weights to produce a filtered texel value.
MAPFILTER_MONO	Supported only on 2D surfaces. This filter is only supported with the monochrome (MONO8) surface format. The monochrome texel block of the specified size surrounding the pixel is selected and filtered.

2.2.2.1 MAPFILTER_NEAREST

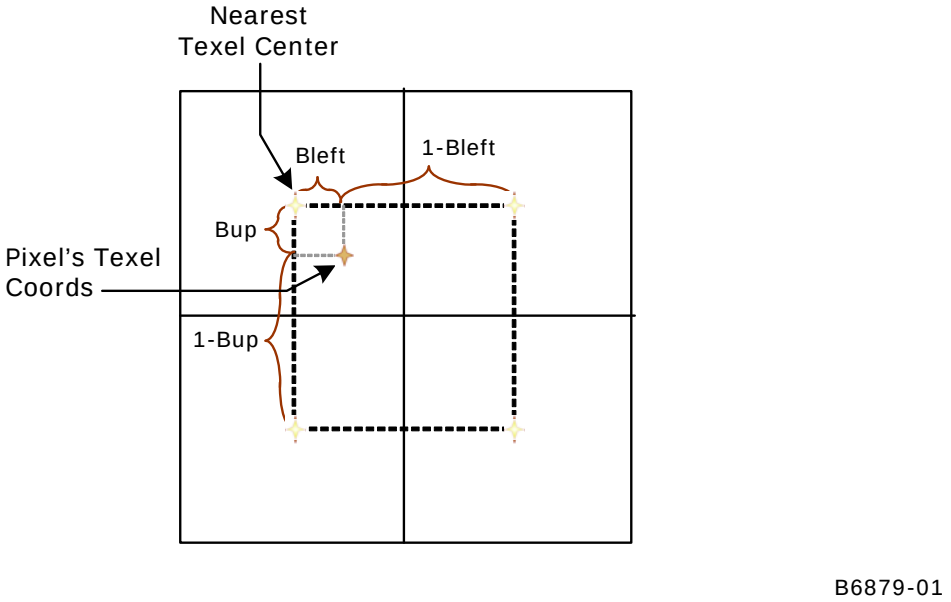
When the MAPFILTER_NEAREST is selected, the texel with coordinates nearest to the pixel's texture coordinate is selected and output as the single texel sample coordinates for the level.

2.2.2.2 MAPFILTER_LINEAR

The following description indicates behavior of the MIPFILTER_LINEAR filter for 2D and CUBE surfaces. 1D and 3D surfaces follow a similar method but with a different number of dimensions available.

When the MAPFILTER_LINEAR filter is selected on a 2D surface, the 2x2 region of texels surrounding the pixel's texture coordinate are sampled and later bilinearly filtered.

Figure 2-3. Bilinear Filter Sampling

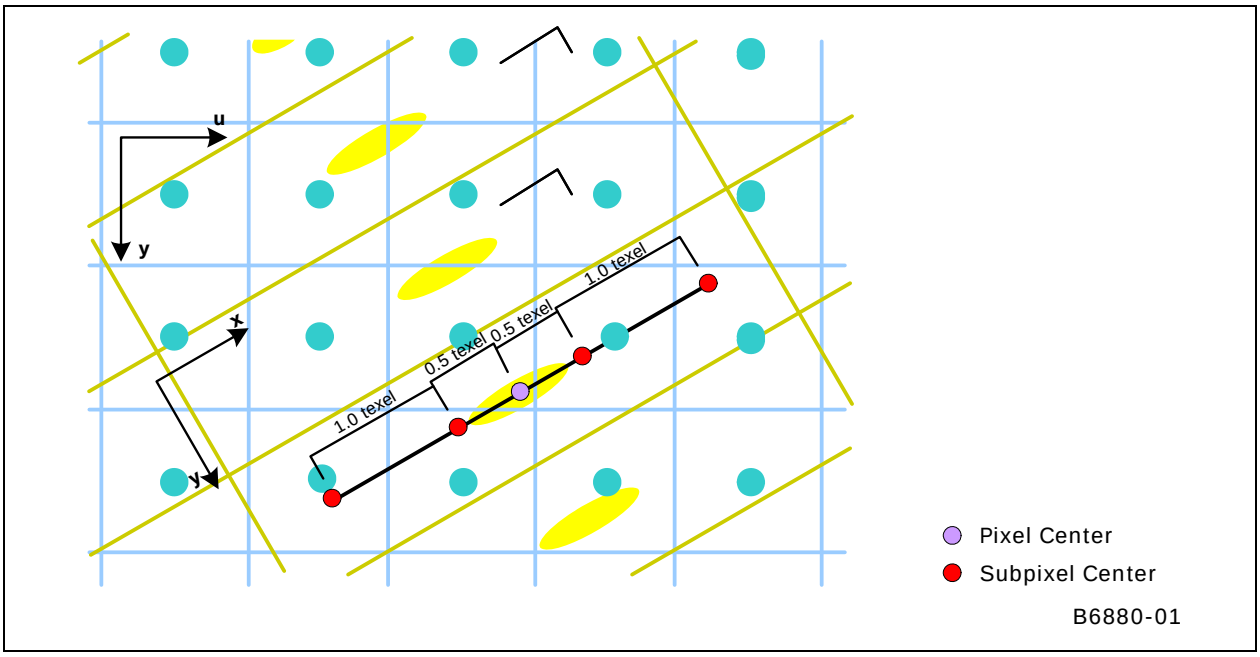


The four texels surrounding the pixel center are chosen for the bilinear filter. The filter weights each texel's contribution according to its distance from the pixel center. Texels further from the pixel center receive a smaller weight.

2.2.2.3 MAPFILTER_ANISOTROPIC

The MAPFILTER_ANISOTROPIC texture filter attempts to compensate for the anisotropic mapping of pixels into texture map space. A possibly non-square set of texel sample locations will be sampled and later filtered. The *MaxAnisotropy* state variable is used to select the maximum aspect ratio of the filter employed, up to 16:1.

The algorithm employed first computes the major and minor axes of the pixel projection onto the texture map. LOD is chosen based on the minor axis length in texel space. The anisotropic "ratio" is equal to the ratio between the major axis length and the minor axis length. The next larger even integer above the ratio determines the anisotropic number of "ways", which determines how many subpixels are chosen. A line along the major axis is determined, and "subpixels" are chosen along this line, spaced one texel apart, as shown in the diagram below. In this diagram, the texels are shown in light blue, and the pixels are in yellow.



Each subpixel samples a bilinear 2x2 around it just as if it was a single pixel. The result of each subpixel is then blended together using equal weights on all interior subpixels (not including the two endpoint subpixels). The endpoint subpixels have lesser weight, the value of which depends on how close the "ratio" is to the number of "ways". This is done to ensure continuous behavior in animation.

2.2.2.4 MAPFILTER_MONO

When the MAPFILTER_MONO filter is selected, a block of monochrome texels surrounding the pixel sample location are read and filtered using the kernel described below. The size of this block is controlled by **Monochrome Filter Height** and **Width** (referred to here as N_v and N_u, respectively) state. Filters from 1x1 to 7x7 are supported (not necessarily square).

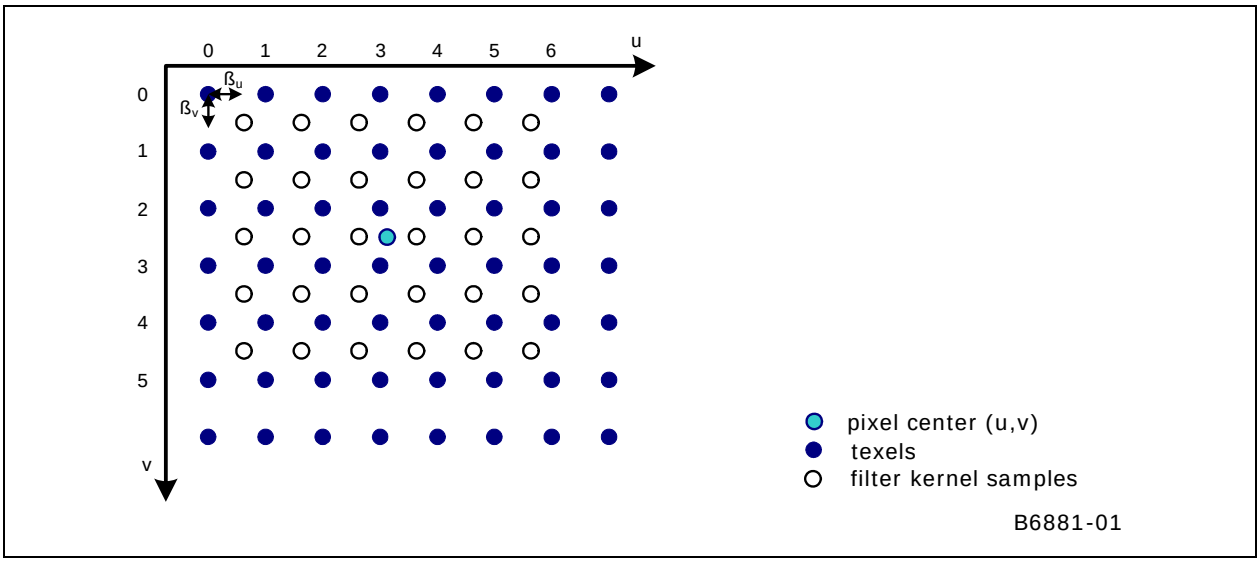
The figure below shows a 6x5 filter kernel as an example. The footprint of the filter (filter kernel samples) is equal to the size of the filter and the pixel center lies at the exact center of this footprint. The position of the upper left filter kernel sample (u_f, v_f) relative to the pixel center at (u, v) is given by the following:

$$u_f = u - \frac{N_u}{2}$$

$$v_f = v - \frac{N_v}{2}$$

β_u and β_v are the fractional parts of u_f and v_f, respectively. The integer parts select the upper left texel for the kernel filter, given here as $T_{0,0}$.

Figure 2-4. Sampling Using MAPFILTER_MONO



The formula for the final filter output F is given by the following. Since this is a monochrome filter, each texel value (T) is a single bit, and the output F is an intensity value that is replicated across the color and alpha channels.

$$S = \frac{1}{N_u * N_v}$$

$$F = \left[(1-\beta_u)(1-\beta_v)\sum_{i=0}^{N_u-1}\sum_{j=0}^{N_v-1} T_{i,j} + \beta_u(1-\beta_v)\sum_{i=1}^{N_u}\sum_{j=0}^{N_v-1} T_{i,j} + (1-\beta_u)\beta_v\sum_{i=0}^{N_u-1}\sum_{j=1}^{N_v} T_{i,j} + \beta_u\beta_v\sum_{i=1}^{N_u}\sum_{j=1}^{N_v} T_{i,j} \right] * S$$

2.2.3 Texture Address Control

The *[TCX,TCY,TCZ]ControlMode* state variables control the access and/or generation of texel data when the specific texture coordinate component falls <u>outside</u> of the normalized texture map coordinate range [0,1).

Note: For **Wrap Shortest** mode, the setup kernel has already taken care of correctly interpolating the texture coordinates. Software will need to specify TEXCOORDMODE_WRAP mode for the sampler that is provided with wrap-shortest texture coordinates, or artifacts may be generated along map edges.

TC[X,Y,Z] Control	Operation
TEXCOORDMODE_CLAMP	Clamp to the texel value at the edge of the map.
TEXCOORDMODE_WRAP	Upon crossing an edge of the map, repeat at the other side of the map in the same dimension.
TEXCOORDMODE_CUBE	Only used for cube maps. Here texels from adjacent cube faces can be sampled along the edges of faces. This is considered the highest quality mode for cube environment maps.
TEXCOORDMODE_MIRROR	Similar to the wrap mode, though reverse direction through the map each time an edge is crossed. INVALID for use with unnormalized

TC[X,Y,Z] Control	Operation
	texture coordinates.
TEXCOORDMODE_MIRROR_ONCE	Similar to the wrap mode, though reverse direction through the map each time an edge is crossed. INVALID for use with unnormalized texture coordinates.

Separate controls are provided for texture TCX, TCY, TCZ coordinate components so, for example, the TCX coordinate can be wrapped while the TCY coordinate is clamped. Note that there are no controls provided for the TCW component as it is only used to scale the other 3 components before addressing modes are applied.

Maximum Wraps/Mirrors

The number of map wraps on a given object is limited to 32. Going beyond this limit is legal, but may result in artifacts due to insufficient internal precision, especially evident with larger surfaces. Precision loss starts at the subtexel level (slight color inaccuracies) and eventually reaches the texel level (choosing the wrong texels for filtering).

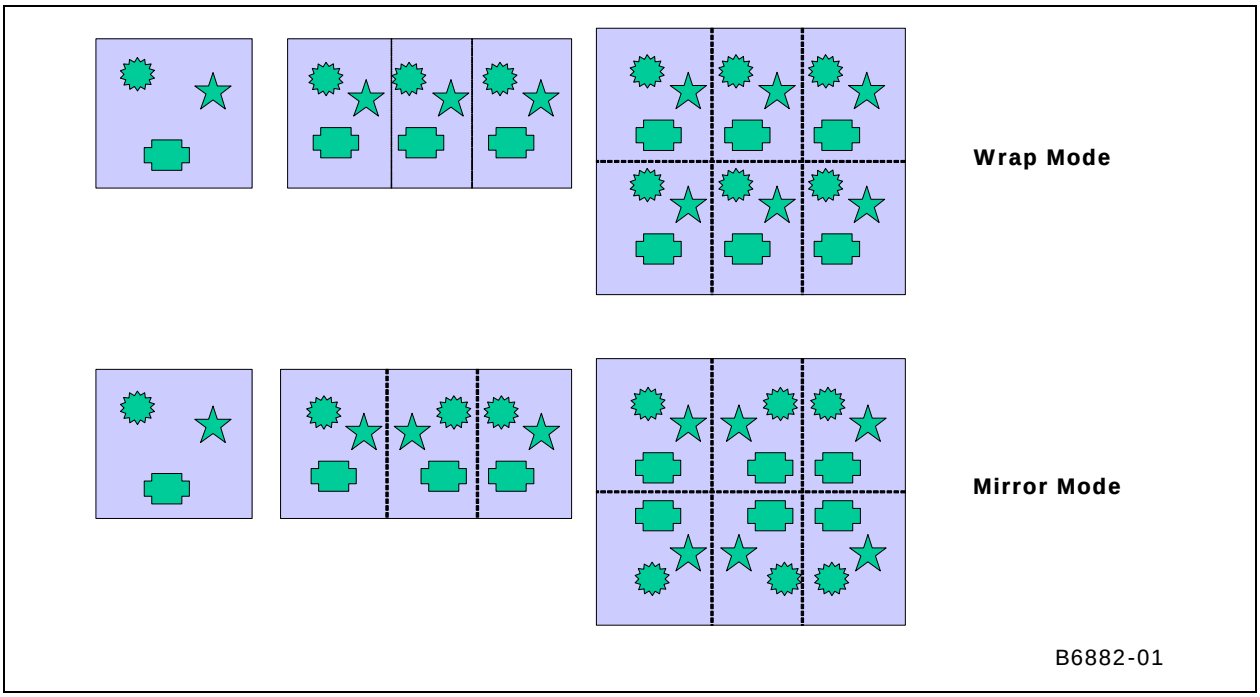
2.2.3.1 TEXCOORDMODE_WRAP Mode

In TEXCOORDMODE_WRAP addressing mode, the integer part of the texture coordinate is discarded, leaving only a fractional coordinate value. This results in the effect of the base map ([0,1)) being continuously repeated in all (axes-aligned) directions. Note that the interpolation between coordinate values 0.1 and 0.9 passes through 0.5 (as opposed to WrapShortest mode which interpolates through 0.0).

2.2.3.2 TEXCOORDMODE_MIRROR Mode

TEXCOORDMODE_MIRROR addressing mode is similar to Wrap mode, though here the base map is flipped at every integer junction. For example, for U values between 0 and 1, the texture is addressed normally, between 1 and 2 the texture is flipped (mirrored), between 2 and 3 the texture is normal again, and so on. The second row of pictures in the figure below indicate a map that is mirrored in one direction and then both directions. You can see that in the mirror mode every other integer map wrap the base map is mirrored in either direction.

Figure 2-5. Texture Wrap vs. Mirror Addressing Mode



2.2.3.3 TEXCOORDMODE_MIRROR_ONCE Mode

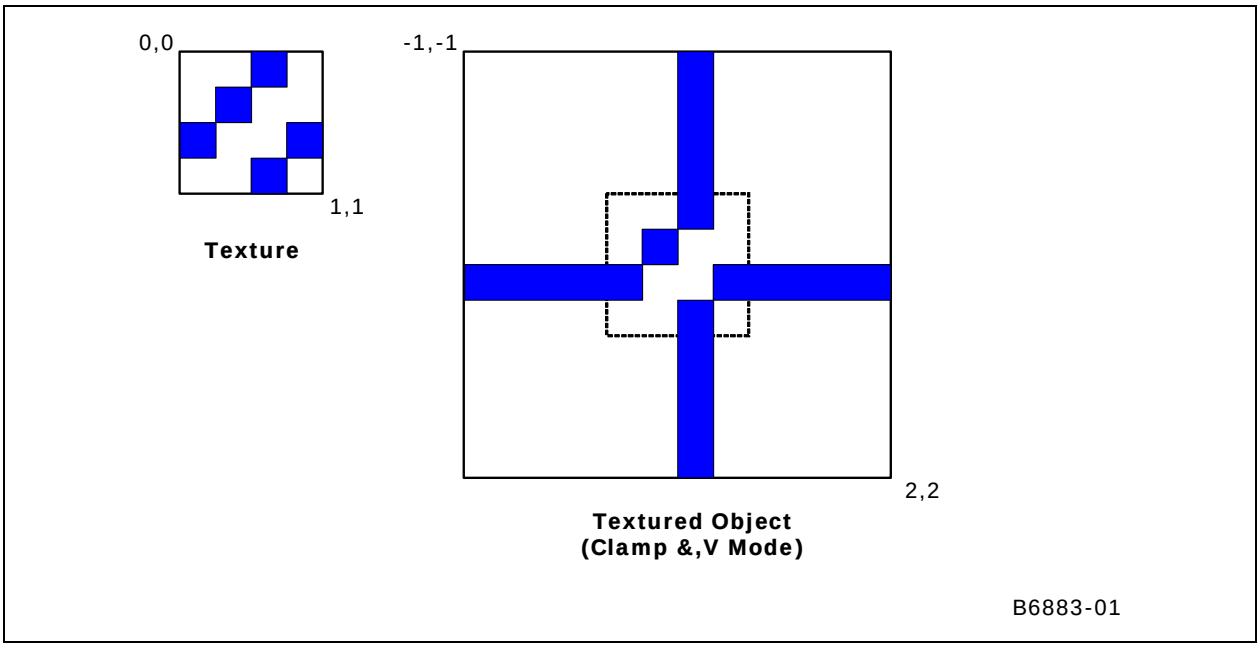
The TEXCOORDMODE_MIRROR_ONCE addressing mode is a combination of Mirror and Clamp modes. The absolute value of the texture coordinate component is first taken (thus mirroring about 0), and then the result is clamped to 1.0. The map is therefore mirrored once about the origin, and then clamped thereafter. This mode is used to reduce the storage required for symmetric maps.

2.2.3.4 TEXCOORDMODE_CLAMP Mode

The TEXCOORDMODE_CLAMP addressing mode repeats the “edge” texel when the texture coordinate extends outside the [0,1) range of the base texture map. This is contrasted to TEXCOORDMODE_CLAMPBORDER mode which defines a separate texel value for off-map samples. TEXCOORDMODE_CLAMP is also supported for cube maps, where texture samples will only be obtained from the intersecting face (even along edges).

The figure below illustrates the effect of clamp mode. The base texture map is shown, along with a texture mapped object with texture coordinates extending outside of the base map region.

Figure 2-6. Texture Clamp Mode



2.2.3.5 TEXCOORDMODE_CUBE Mode

For cube map textures TEXCOORDMODE_CUBE addressing mode can be set to allow inter-face filtering. When texel sample coordinates that extend beyond the selected cube face (e.g., due to intra-level filtering near a cube edge), the correct sample coordinates on the adjoining face will be computed. This will eliminate artifacts along the cube edges, though some artifacts at cube corners may still be present.

2.3 Texel Fetch

The Texel Fetch function of the Sampling Engine reads the texture map contents specified by the texture addresses associated with each texel sample. The texture data is read either directly from the memory-resident texture map, or from internal texture caches. The texture caches can be invalidated by the **Sampler Cache Invalidate** field of the MI_FLUSH instruction or via the **Read Cache Flush Enable** bit of PIPE_CONTROL. Except for consideration of coherency with CPU writes to textures and rendered textures, the texture cache does not affect the functional operation of the Sampling Engine pipeline.

When the surface format of a texture is defined as being a compressed surface, the Sampler will automatically decompress from the stored format into the appropriate [A]RGB values. The compressed texture storage formats and decompression algorithms can be found in the *Memory Data Formats* chapter. When the surface format of a texture is defined as being an index into the texture palette (format names includiong "Px"), the palette lookup of the index determines the appropriate RGB values.

2.3.1 Texel Chroma Keying

ChromaKey is a term used to describe a method of effectively removing or replacing a specific range of texel values from a map that is applied to a primitive, e.g., in order to define transparent regions in an RGB map. The Texel Chroma Keying function of the Sampling Engine pipeline conditionally tests texel samples against a "key" range, and takes certain actions if any texel samples are found to match the key.

2.3.1.1 Chroma Key Testing

ChromaKey refers to testing the texel sample components to see if they fall within a range of texel values, as defined by *ChromaKey[][High,Low]* state variables. If each component of a texel sample is found to lie within the respective (inclusive) range and ChromaKey is enabled, then an action will be taken to remove this contribution to the resulting texel stream output. Comparison is done separately on each of the channels and only if all 4 channels are within range the texel will be eliminated.

The Chroma Keying function is enabled on a per-sampler basis by the *ChromaKeyEnable* state variable.

The *ChromaKey[][High,Low]* state variables define the tested color range for a particular texture map.

2.3.1.2 Chroma Key Effects

There are two operations that can be performed to "remove" matching texel samples from the image. The *ChromaKeyEnable* state variable must first enable the chroma key function. The *ChromaKeyMode* state variable then specifies which operation to perform on a per-sampler basis.

The *ChromaKeyMode* state variable has the following two possible values:

KEYFILTER_KILL_ON_ANY_MATCH: Kill the pixel if any contributing texel sample matches the key

KEYFILTER_REPLACE_BLACK: Here the sample is replaced with (0,0,0,0). This matches the Direct3D COLORKEYBLENDENABLE functionality

The Kill Pixel operation has an effect on a pixel only if the associated sampler is referenced by a sample instruction in the pixel shader program. If the sampler is not referenced, the chroma key compare is not done and pixels cannot be killed based on it.

2.4 Shadow Prefilter Compare

When a *sample_c* message type is processed, a special shadow-mapping precomparison is performed on the texture sample values prior to filtering. Specifically, each texture sample value is compared to the "ref" component of the input message, using a compare function selected by *ShadowFunction*, and described in the table below. Note that only single-channel texel formats are supported for shadow mapping, and so there is no specific color channel on which the comparison occurs.

ShadowFunction	Result
PREFILTEROP_ALWAYS	0.0
PREFILTEROP_NEVER	1.0

ShadowFunction	Result
PREFILTEROP_LESS	(texel < ref) ? 0.0 : 1.0
PREFILTEROP_EQUAL	(texel == ref) ? 0.0 : 1.0
PREFILTEROP_LEQUAL	(texel <= ref) ? 0.0 : 1.0
PREFILTEROP_GREATER	(texel > ref) ? 0.0 : 1.0
PREFILTEROP_NOTEQUAL	(texel != ref) ? 0.0 : 1.0
PREFILTEROP_GEQUAL	(texel >= ref) ? 0.0 : 1.0

The binary result of each comparison is fed into the subsequent texture filter operation (in place of the texel's value which would normally be used).

Software is responsible for programming the "ref" component of the input message such that it approximates the same distance metric programmed in the texture map (e.g., distance from a specific light to the object pixel). In this way, the comparison function can be used to generate "in shadow" status for each texture sample, and the filtering operation can be used to provide soft shadow edges.

Programming Note:

- Refer to the Surface Formats table in section 0 for the specific surface formats that are supported with shadow mapping.

2.5 Texel Filtering

The Texel Filtering function of the Sampling Engine performs any required filtering of multiple texel values on and possibly between texture map layers and levels. The output of this function is a single texel color value.

The state variables *MinFilter*, *MagFilter*, and *MipFilter* are used to control the filtering of texel values. The *MipFilter* state variable specifies how many mipmap levels are included in the filter, and how the results of any filtering on these separate levels are combined to produce a final texel color. The *MinFilter* and *MagFilter* state variables specify how texel samples are filtered within a level.

2.6 Texel Color Gamma Linearization

This function is supported to allow pre-gamma-corrected texel RGB (not A) colors to be mapped back into linear (gamma=1.0) gamma space prior to (possible) blending with, and writing to the Color Buffer. This permits higher quality image blending by performing the blending on colors in linear gamma space.

This function is enabled on a per-texture basis by use of a surface format with "_SRGB" in its name. If enabled, the pre-filtered texel RGB color to be converted from gamma=2.4 space to gamma=1.0 space by applying a ^(1/2.4) = ^0.4167 exponential function.

2.7 Multisampled Surface Behavior [DevSNB+]

The ld message has added an additional parameter for sample index (si) to support unfiltered loading from a multisampled surface.

The sampleinfo message returns specific parameters associated with a multisample surface. The resinfo message returns the height, width, depth, and MIP count of the surface (in units of *pixels*, not samples).

Any of the other messages (sample*, LOD, load4) used with a (4x) multisampled surface will in-effect sample a surface with double the height and width as that indicated in the surface state. Each pixel position on the original-sized surface is replaced with a 2x2 of samples with the following arrangement:

sample 0	sample 2
sample 1	sample 3

This behavior is useful to implement the multisample resolve operation by selecting MAPFILTER_LINEAR and rendering a full-screen rectangle half the size in each dimension of the source texture map (multisampled surface). If pixel offsets are set correctly, each pixel is the average of the four underlying samples.

2.8 Denoise/Deinterlacer [DevSNB]

The Denoise/Deinterlacer function takes a 4:2:0 or 4:2:2 video stream and first apply a denoise filter to it and then deinterlace it.

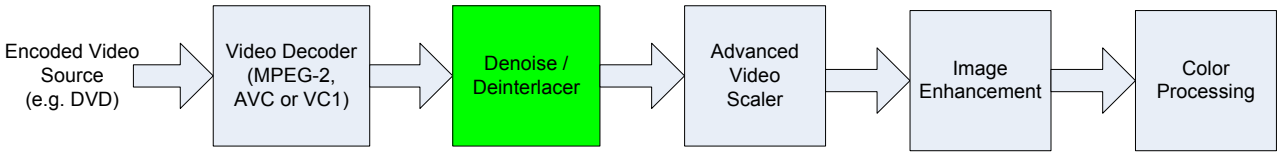
The denoise filter is applied before the deinterlacer. The denoise filter detects and tries to minimize noise in the input field, while the deinterlacer takes a field consisting of every other lines converts a field into a frame. This block also gathers statistics for a global noise estimate made in software at the end of the frame which is used in following frames to tune the denoise filter and image enhancement filter.

The deinterlacer takes the top and bottom fields of each frame and converts them into two individual frames. This block also gathers statistics for a film mode detector in software run at the end of the frame. If the film mode detector for the previous frame concludes that the input is progressive rather than interlaced then the fields will be put together in the best order rather than being interlaced.

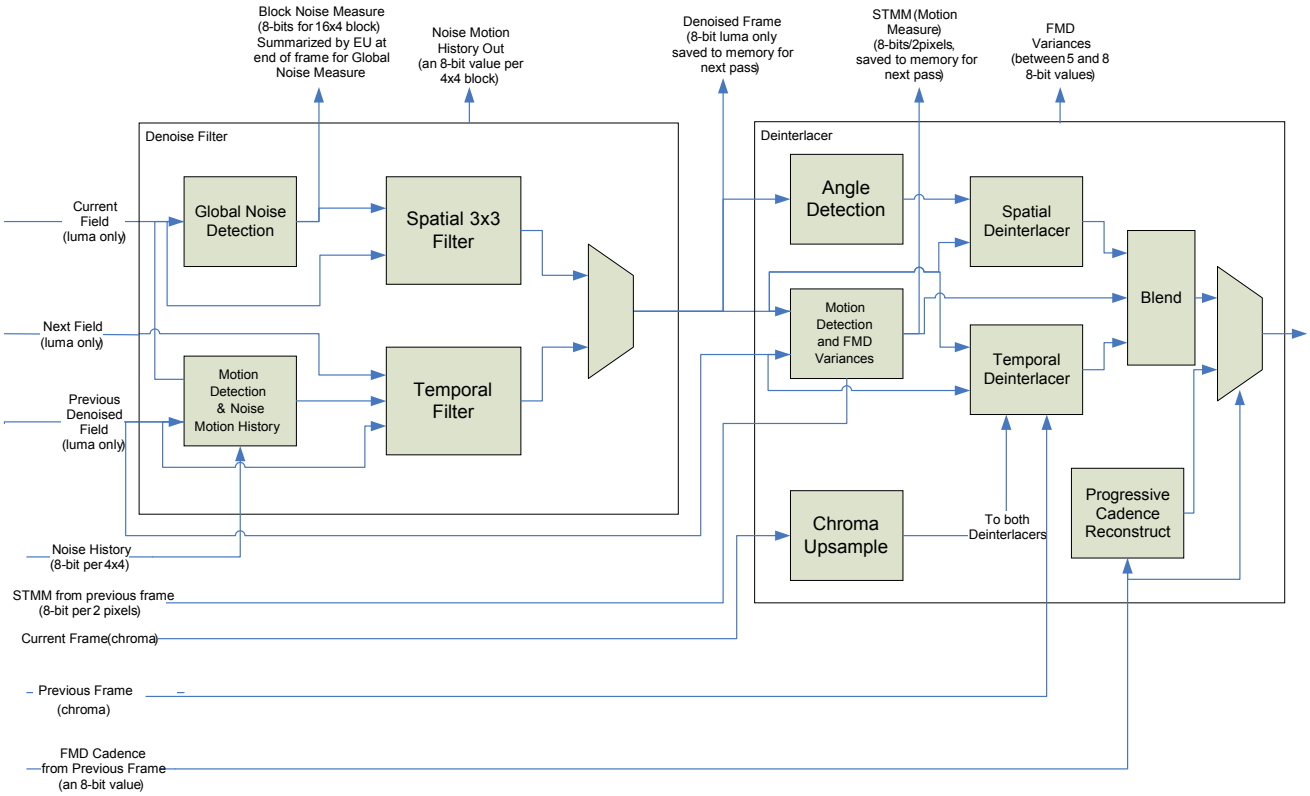
2.8.1 Introduction

2.8.1.1 Overview

This diagram shows how the Denoise/Deinterlacer fits in with the other functions of the video pipe. This is only one possible usage model, other models are possible.



2.8.1.2 Block Diagram



2.8.1.3 Features

- **Denoise Filter** – detects noise and motion and filters the block with either a temporal filter when little motion is detected or a spatial filter. Noise estimates are kept between frames and blended together. Since the filter is before the deinterlacer it works on individual fields rather than frames. This usually improves the operation since the deinterlacer can take a single pixel of noise and spread it to an adjacent pixel, making it harder to remove. The denoise filter works the same whether deinterlacing or progressive cadence reconstruction is being done.

- **Block Noise Estimate** (BNE) – part of the Global Noise Estimate (GNE) algorithm, this estimates the noise over the entire block. The GNE will be calculated at the end of the frame by combining all the BNEs. The final GNE value is used to control the denoise filter for the next frame.

- **Film Mode Detection** (FMD) Variances – FMD determines if the input fields were created by sampling film and converting it to interlaced video. If so the deinterlacer is turned off in favor of reconstructing the frame from adjacent fields. Various sum-of-absolute differences are calcluated per block. The FMD algorithm is run at the end of the frame by looking at the variances of all blocks for both fields in the frame.

- **Deinterlacer** – Estimates how much motion is occuring across the fields. Low motion scenes are reconstructed by averaging pixels from fields from nearby times (temporal deinterlacer), while high motion scenes are reconstructed by interpolating pixels from nearby space (spatial deinterlacer).

- **Progressive Cadence Reconstruction** – If the FMD for the previous frame determines that film was converted into interlaced video, then this block reconstructs the original frame by directly putting together adjacent fields.

- **Chroma Upsampling** – If the input is 4:2:0 then chroma will be doubled vertically to convert to 4:2:2. Chroma will then either go through it's own version of the deinterlacer or progressive cadence reconstruction.

When DI is enabled, the output for a 16x4 block is sent to the EU for further processing and writing to memory. When DI is disabled and DN enabled the output for a 16x8 block is sent to the EU.

Formats supported are:

NV12 is supported for hardware video decode.

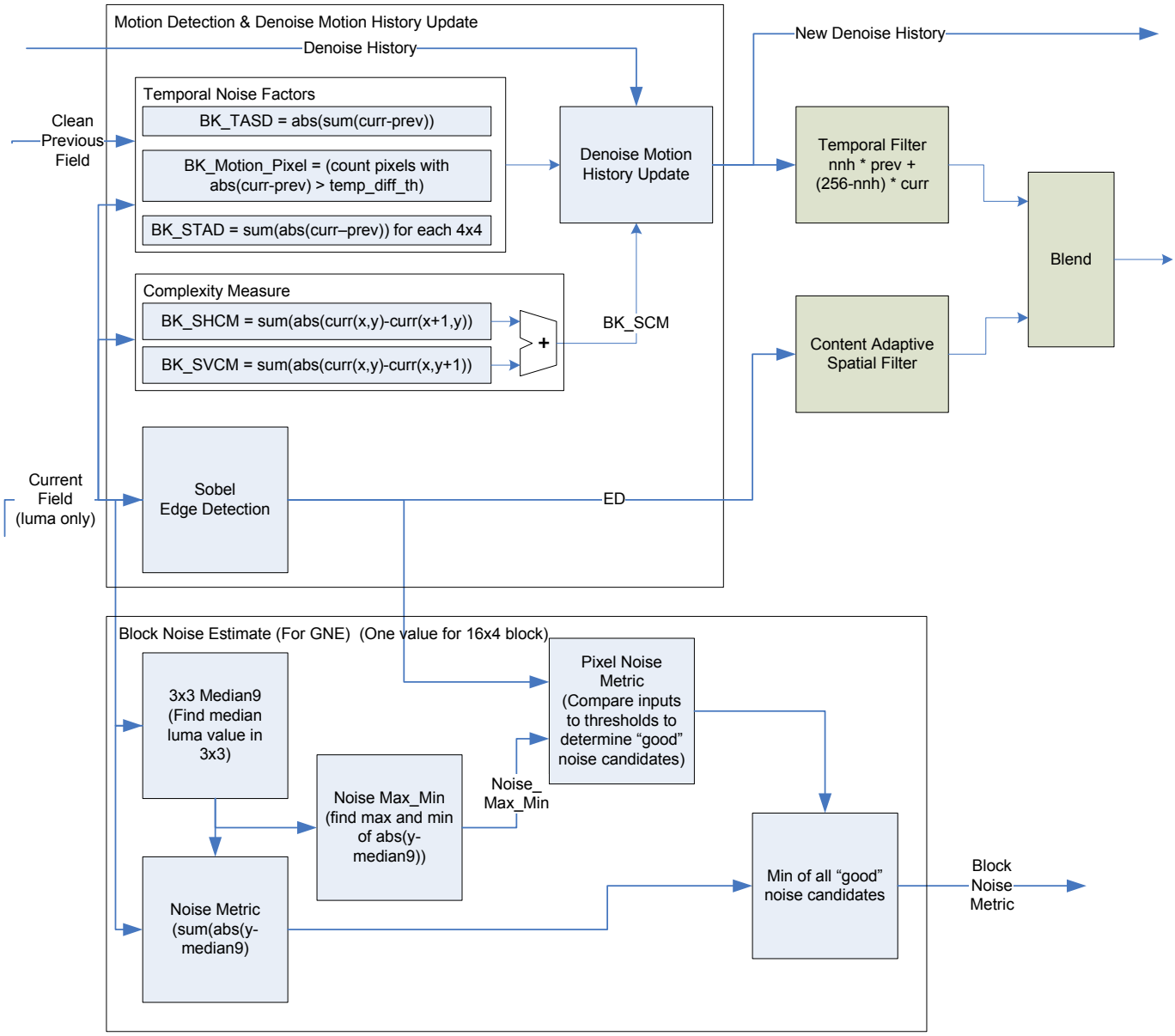
UYVY, YUY2 and NV12 are required for WHQL.

YV12 and I420 are supported for software video decode.

IMC3 and IMC4 are supported as internal temporary formats.

NV11 and P208 are not supported, since they have been removed from the WHQL logo requirement.

2.8.2 Denoise Algorithm



2.8.2.1 Motion Detection and Noise History Update

This block detection motion for the denoise filter, which it then combines with motion detected in the past in the same part of the screen. The Denoise History is both saved to memory and also used to control the temporal denoise filter.

The block calculates a number of values for updating the Denoise History. One value is calculated per 4x4 block (pixels from both fields, interleaved):

Block Sum of Temporal Absolute Difference:

BK_STAD = $\sum_{x=0}^{3}\sum_{y=0}^{3} abs(curr(x,y) - prev(x,y))$

Where curr(x,y) and prev(x,y) are lumas from the current and previous field. The previous field should have already been run through the denoise filter.

Count of motion pixels: increment BK_Motion_Pixel for every pixel in the 4x4 for which: (abs(curr(x,y) – prev(x,y)) >= temporal_diff_th.

Absolute Sum of Temporal Difference sums the differences without the initial absolute value, so that random motions will tend to cancel out:

BK_TASD = $abs(\sum_{x=0}^{3}\sum_{y=0}^{3}(curr(x,y) - prev(x,y)))$

Sum of Complexity Measure looks for differences in the spatial domain:

BK_SHCM = $\sum_{x=0}^{2}\sum_{y=0}^{3} abs(curr(x,y) - curr(x+1,y))$ // sum of 12 pixel pairs

BK_SVCM = $\sum_{x=0}^{3}\sum_{y=0}^{2} abs(curr(x,y) - curr(x,y+1))$ // sum of 12 pixel pairs

BK_SCM = BK_SHCM + BK_SVCM

Denoise Motion History Update (for an 8-bit motion history):

```
if (BK_STAD>=dnmh_stad_th) or (BK_Motion_Pixel > dnmh_mp_th) { // Motion Block

    motion_block = 1;

    if (denoise_history >= 128)

            new_denoise_history = denoise_history / 2;

    else

            new_denoise_history = 0;

} else {   // static block

    motion_block = 0;

    if  (denoise_history < 128)

            new_denoise_history = 128;
```

```
    else if (denoise_history < dnmh_history_max)

            new_denoise_history = denoise_history + dnmh_delta;  // default value 8 for delta

    else

            new_denoise_history = denoise_history;

    if ((BK_TASD > dnmh_tasd_th) and (BK_SCM < dnmh_scm_th))

            new_denoise_history = 128;

}
```

2.8.2.2 Temporal Filter

For each pixel we need to filter we look at the noise history for the associated 4x4.

temporal_denoised = (new_denoise_history * curr(x,y) + (256 – new_denoise_history) * prev(x,y) +128) >> 8

2.8.2.3 Context Adaptive Spatial Filter

For each pixel in the local 3x3, compare it's luma to the lumas of the pixel to be filtered. Each pixel for which the absolute difference is less than good_neighbor_th (see state variable in section 2.11.3.2) is marked as a "good neighbor":

The filtered pixel is then equal to:

spatial_denoised = ∑ Good_neighbor luma / num_good_neighbors

The divide is implemented as a multiply by a table lookup:

spatial_denoised = ((∑Good_neighbor luma + (num_good_neighbors >>1)) * gn_q_table[num_good_neighbors-1]) >> 11

Note: The number of good neighbors varies from 1 to 9 since the center pixel is always good. Gn_q_table provides the reciprocal:

```
    gn_q_table[9] = {2048, 1024, 682, 512, 409, 341, 292, 256, 227};
```

2.8.2.4 Denoise Blend

The denoise blend combines the temporal and spatial denoise outputs.

First we check to see if the temporal is out of the local range, if so we use the average of the denoised and the local limit instead:

```
if (temporal_denoised >= block_max)

     temporal_denoised=(temporal_denoised+block_max)>>1;
```

```
if (temporal_denoised < block_min)
    temporal_denoised=(temporal_denoised+block_min)>>1;
```

Where block_max and block_min are the largest and smallest luma values in the local 3x3 (can be shared with BNE calculation).

Next we decide between using the spatial and temporal denoise output:

```
t_diff = abs(curr(x,y) – prev(x,y);
if (t_diff < temporal_diff_th) {
    if (motion_block==1)
        denoise_out = spatial_denoised;
    else {
        if (t_diff < temp_diff_low)
            denoise_out=temporal_denoised;
        else {
            denoise_out=
              (spatial_denoised*(t_diff-temp_diff_low) +
               temporal_denoised*(temporal_diff_th-t_diff)+
               (temporal_diff_th-temp_diff_low)/2
               ) * q_table[temporal_diff_th-temp_diff_low-1]) >> 10;
    }
} else {
    denoise_out = spatial_denoised;
}
```

Motion_block is defined in section 2.8.2.1 above. T_diff can be limited to 6-bits to minimize the multipler gates required in the blend. A divide is eliminated by providing the reciprocal of the divisor in the q_table which is defined:

```
q_table[16] = {1024,512,341,256,205,171,146,128,114,102,93,85,79,73,68,64}
```

The following restrictions also apply:

1) Temporal_diff_th – temp_diff_low is limited in the state variable definition to the range 16 to 1.

2) Since t_diff<temporal_diff_th; (t_diff – temp_diff_low) is less than 16

3) Since t_diff>=temp_diff_low; (temporal_diff_th-t_diff) is less then or equal to 16.

The precision needed for spatial_denoised*(t_diff-temp_diff_low) is 8-bit times 4-bits to produce 12-bits. The other multiply is 8 by 5 to produce 13-bits; the extra bit is needed for 16. The multiplier to implement the divide will be a 13-bit times the 11-bit number out of q_table, but this could be reduced by implementing a 13x9 bit multiplier with the top 2 bits controlling a mux since the only table entries that use them are 1024 and 512.

2.8.3 Block Noise Estimate (part of Global Noise Estimate)

Edge detection is done on every pixel in the 16x4 (DI enabled) or 16x8 (DN only) by estimating a gradient on the 3x3 neighborhood of pixels in the current field. The calculation only uses a multiply of 2, so shifts and add are all that is needed. Currently only vertical and horizontal edges are detected, 45 degrees is a potential improvement.

Hrz Edge = abs(c(x-1,y-1) +2*c(x,y-1) +c(x+1,y-1) –c(x-1,y+1) –2*c(x,y+1) –c(x+1,y+1))

Vrt Edge = abs(c(x-1,y-1) +2*c(x-1,y) +c(x-1,y+1) –c(x+1,y-1) –2*c(x+1,y) –c(x+1,y+1))

The Hrz_Edge and Vrt_Edge are added together and if the sum is greater than bne_edge_th then an edge is detected:

ED = (Hrz_Edge +Vrt_Edge) >> 3

- median9 – the median of the 9 luma values for the 3x3 neighborhood pixels is used. Median5, the median of the pixels above/below/right/left/center may be satisfactory as a lower gate count solution.
- for each pixel luma “y” in 3x3: noise_metric = sum(y – median9)
- noise_min = min(abs(y-median9)) - min of all 9 ys in 3x3
- noise_max = max(abs(y-median9)) – max of all 9 ys in 3x3
- noise_min_max = noise_max(x,y) – noise_min(x,y)
- pixel_noise_metric = noise_metric if (ED(x,y) < bne_edge_th) and (noise_max_min(x,y) < bne_nn_th) block_noise_estimate = min of all pixel_noise_metrics that pass the if test in the 16x4 (use 255 if no pixels pass the test)

If the block_noise_estimate is less than 255 then it is added to a sum gathered across the entire frame. The summation will need to be 23-bits wide to be able to sum 8-bit values for all 32,400 blocks in a 1920x1080 frame. In addition, there will be a count of the number of blocks in the sum. The data will be written to memory at the end of the frame. Two sets of counters are needed to support 2 simultaneous streams. The streams are distinguished by the dndi_stream_id state variable in the DI state.

The per block block_noise_estimate is also sent to the EU in the output message for possible use by the video encoder.

2.8.4 Deinterlacer Algorithm

The overall goal of the motion adaptive deinterlacer is to convert an interlaced video stream made of fields of alternating lines into a progressive video stream made of frames in which every line is provided.

If there is no motion in a scene, then the missing lines can be provided by looking at the previous or next fields, both of which have the missing lines. If there is a great deal of motion in the scene, then objects in the previous and next fields will have moved, so we can't use them for the missing pixels. Instead we have to interpolate from the neighboring lines to fill in the missing pixels. This can be thought of as interpolating in time if there is no motion and interpolating in space if there is motion.

This idea is implemented by creating a measure of motion on a per 2 pixel basis called the Spatial-Temporal Motion Measure (STMM). If this measure shows that there is little motion in an area around the pixels, then the missing pixels are created by averaging the pixel values from the previous and next frame. If the STMM shows that there is motion, then the missing pixels are filled in by interpolating from neighboring lines with the Spatial Deinterlacer (SDI). The two different ways to interpolate the missing pixels are blended for intermediate values of STMM to prevent sudden transitions.

The Deinterlacer uses two frames for reference. The current frame contains the field that we are deinterlacing. The reference frame is the closest frame in time to the field that we are deinterlacing – if we are working on the 1st field then it is the previous frame, if it is the 2nd field then it is the next frame.

2.8.4.1 Spatial-Temporal Motion Measure

This algorithm combines a complexity measure with a estimate of motion. This prevents high complexity scenes from incorrectly causing motion to be detected. It is calculated for a set of pixels 2 wide by 1 high.

Complexity is measured in the vertical and horizontal directions with the SVCM and SHCM. For each set of 2 pixels which need to be interpolated, a window of pixels is used that is 4 wide and 5 high - +/-1 pixel in X and +/- 2 pixels in Y. The pixels values are taken from both the current and previous field - for example, if we are deinterlacing the top field then lines y+2,y, and y-2 will come from the top field; while line y+1 and y-1 will come from the bottom field.

Spatial vertical complexity measure (SVCM) is a sum of all the differences in the vertical direction for a window around the current pixels. If we take x,y=0,0 as the left pixel of our 2x1 then:

$$\text{SVCM} = \sum_{x=0}^{1}\sum_{y=0}^{2} abs(c(x,y) - c(x,y-2))$$

Where c(x,y) is the luma value at that x,y location in the current frame. Note that we are skipping by 2 in the Y direction to ensure that the compares are only done with lines from the same field.

Spatial horizontal complexity measure (SHCM) is a sum of differences in the horizontal direction.

$$\text{SHCM} = \sum_{x=-1}^{1}\sum_{y=-1}^{y=1} abs(c(x,y) - c(x+1,y))$$

The vertical edge complexity measure (VECM) is a sum of difference in the horizontal direction similar to SHCM, but uses different pixels from the window.

VECM = $\left(\left(\sum_{y=-2}^{y=2} abs(c(x,y)-c(x+1,y))\right) * \text{vecm_mul}\right)$ >>5

Temporal Difference Measure (TDM) is a measure of differences between pairs of fields with the same lines. It uses filtered versions of c(x,y) from the current frame and r(x,y) from the reference frame (either the previous or next frame).

The filter used is a cross filter which uses the pixels above, below, to the right and to the left of the needed pixel in the same field. When denoise filter is enabled, the filter input c(x,y) is a denoised pixel only if -2<=y<=6 for dndi_topfirst=1, and -3<=Y<=5 for dndi_topfirst=0. Note that r(x,y) is a denoised pixel regardless of y.

c'(x,y) = (2*c(x,y) +c(x-1,y) +c(x+1,y) +2*c(x,y-2) +2*c(x,y+2)) >> 3 (Done for both c(x,y) and r(x,y))

TDM = $\sum_{x=-1}^{2}\sum_{y=-2}^{2} abs(c'(x,y)-r'(x,y))$

STMM is then calculated by :

STMM = ((TDM >>tdm_shift1)<<tdm_shift2) / (SCM >> 4) + stmm_c2)

where SCM = max(0, SVCM+SHCM-VECM). Tdm_shift1 is used to quantize the STMM result, while Tdm_shift2 is used to set the STMM range. Tdm_shift1 can range from 4 to 6; since TDM has 13 bits this results in between 9 and 7 bits of precision. Tdm_shift2 can range from 6 to 8, producing a value between 17 and 13 bits, of which only 9-bits are non-zero. The divide can be implemented by a 8-bit reciprocal table followed by an 9 -bit x 8-bit multiply by the TDM value, which finally produces an output of 8-bits.

STMM is then smoothed with an exponential moving average with the STMM saved from the previous field:

```
if  (STMM > stmm_md_th)

        STMM2 = (stmm_trc1 * STMM_s + (256-stmm_trc1)*STMM) / 256

else

        STMM2 = (stmm_trc2 * STMM_s + (256-stmm_trc2)*STMM) / 256
```

with state variables stmm_trc1 (typical value 64), stmm_trc2 (typical value 200), and stmm_md_th.

This process prevent sudden changes in STMM, though STMM over a certain value uses a smaller smoothing constant (c1) which allows it to change faster. STMM2 is stored to memory to be read as STMM_s by the next frame.

One final step is used to prevent sudden drops in STMM in the horizontal direction – taking the maximum of the STMM on the right and left sides:

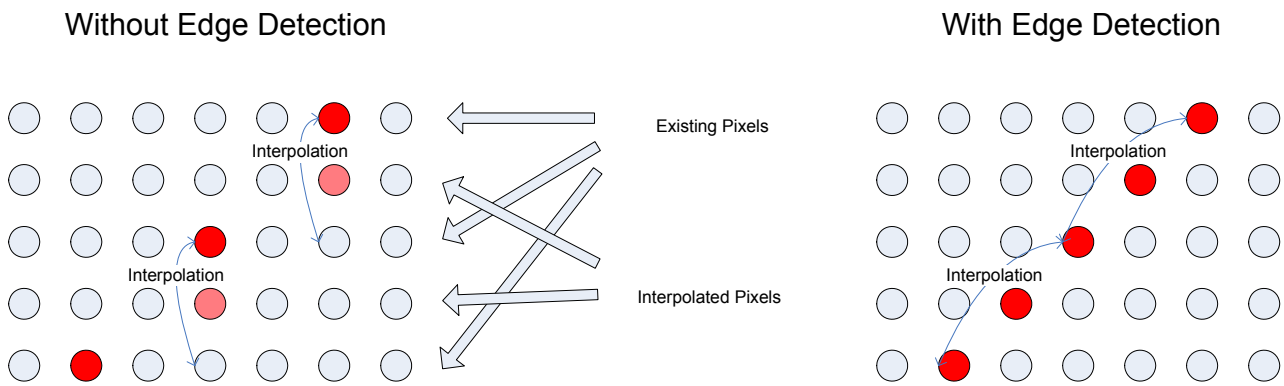
STMM3(x) = max (STMM2(x-2), STMM2(x), STMM2(x+2))

The resulting STMM3 will be used as a blending factor between the spatial and temporal deinterlacer.

2.8.4.2 Spatial Deinterlacer Angle Detection

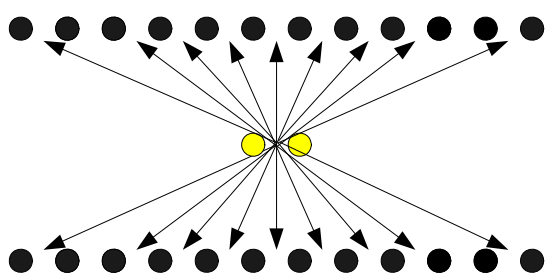
Deciding the best pixels to interpolate in the current field is the job of the spatial deinterlacer. The simplest method would be to interpolate directly from the pixels above and below the missing pixels, but this can look bad; edges and lines particularly look jagged with this solution.

A better solution is to detect the direction of edges in the pixel neighborhood and interpolate along the edge direction.



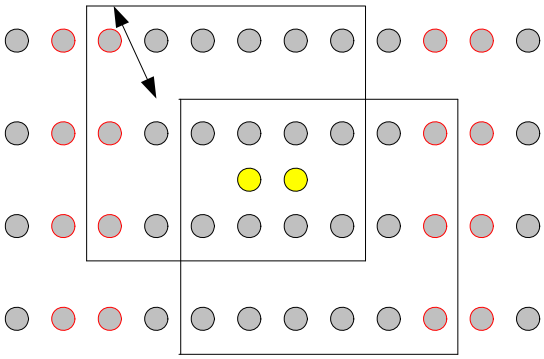
Edge detection is done per 2 pixels to lower the compute needed (may change in this implementation depending on quality). Edge detection is done by taking a window of pixels around the pixels of interest and comparing with a window offset in the direction being tested. The more simularity between the windows the more likely it is that the movement is in the direction of an edge.

We test 9 different directions to pick the best edge: vertical, +/-45°, +/-27°, +/-18° and +/-11 degrees. The window offset for 45° x+/-1, likewise the offset of 27° is x+/-2, 18° is x+/-3, and 11° is x+/-5. X+4 is not used because the gap between 18° and 11° is too small to make it worth checking.



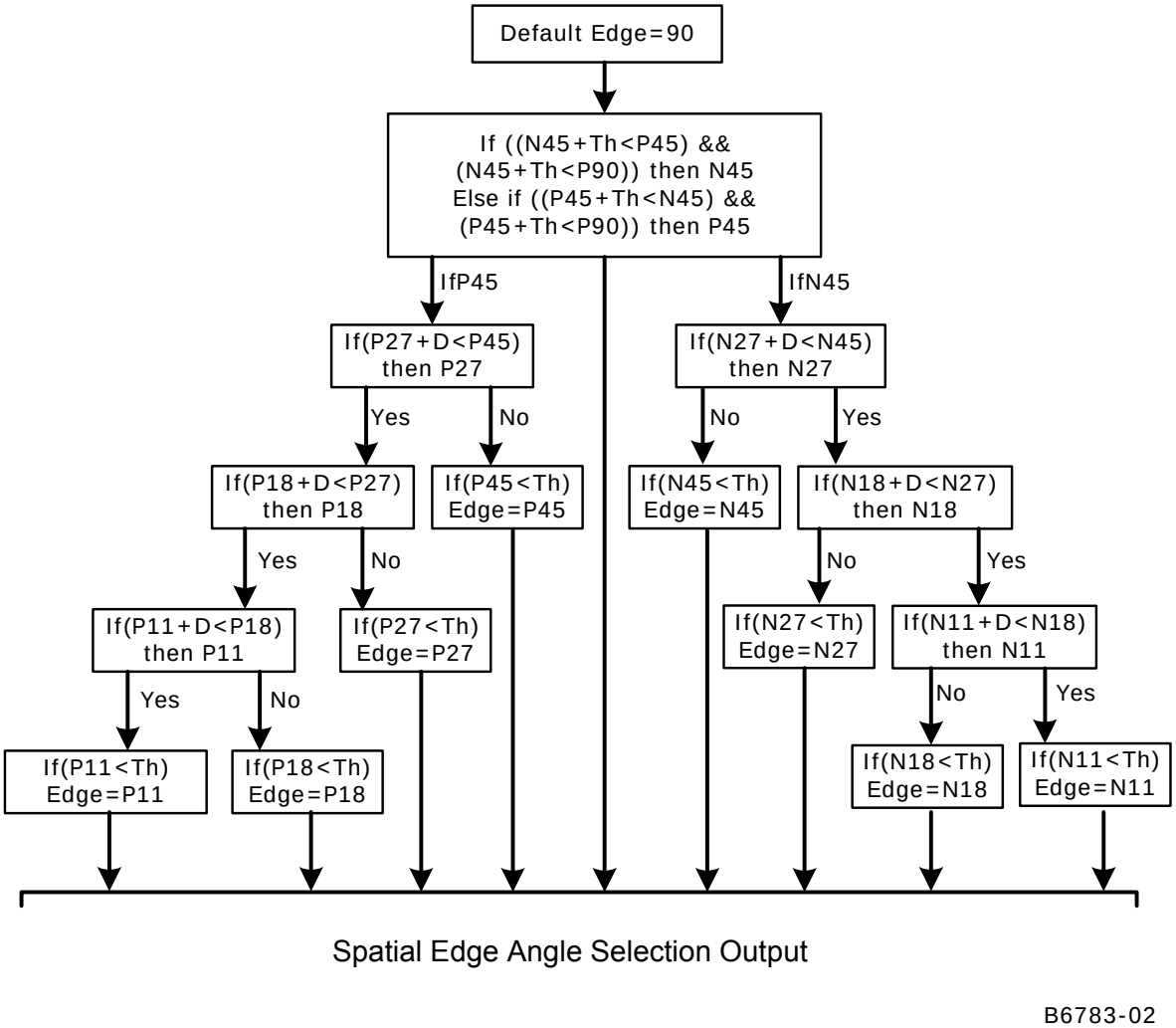
Use x,y=0,0 for the left pixel of the pair that we want to interpolate, and xoffset is the offset described in the above paragraph. The equation for each angle checked is:

$$\text{AngleCost_6x3} = \sum_{x=-2}^{3} \sum_{y=-2,0,2} abs(n(x+xoffset, y+1) - n(x-xoffset, y-1))$$



The above picture illustrates the 45 degree angle compution – taking the sum-of-absolute differences of the two 6x3 blocks around the 2 pixels that need an angle estimated. Each block is offset by 1 in Y and X in opposite direction. The offset in X is larger for the other angles, of course. Angle detection requires up to 7 pixels (offset of 5 plus 2 to get all the pixels in the 6x3) on the right and left of the output block, requiring the input to the deinterlacer from the denoise to be 16 + 7 + 7, or 30 pixels.

Once we have all the angle values, the final decision is done by comparing them with each other. In the following diagram N45 indicates the AngleCost_6x3 for -45°, likewise P27 is the value for +27°, etc. Th and D are constants used to fine tune the algorithm.



Any missing arcs in the above diagram use the default edge of 90 degrees; for example if the lower left box has P11 >= Th then the default will be used.

2.8.4.2.1 Angle Robustness Check

Three special checks are made to eliminate incorrect angle detection.

Fallback Mode 1

Moving regions with fine details can confuse the angle detection. This fallback mode will detect fine details and fall back to 90 degrees if they are detected.

SUM_H1(x,y) = $\sum_{s=-2}^{3} abs(c(x+s,y)-c(x+s+1,y))$

This sum is similar to SHCM, but over a horizontal line of -2 to +3 only.

SUM_H2(x,y) = $\max_{s=-2,-1,\ldots,3}(abs(c(x-2,y)-c(x+s,y))+abs(c(x+s,y)-c(x+4,y))$

if (SUM_H1(y-1) + SUM_H1(y+1) > SUM_H2(y-1) + SUM_H2(y+1) + sdi_t1 &&

SUM_H1(y-1) + SUM_H1(y+1) >= sdi_t2) Then use 90 degree

The final decision for each pixel is done using the sums from above and below the current Y.

Fallback Mode 2

Sometimes the 6x3 angle detection window makes mistakes due to pixels on the edge of the window. Adding a check using a 2x1 window fixes these problems:

If(AngleCost_6x3(90 degree) + (AngleCost_2x1(90 degree)<<3) <

AngleCost_6x3(best angle) + ((AngleCost_2x1(best angle) + sdi_angle2x1)<<3)) then use 90 degree

AngleCost_2x1 is the same as AngleCost_6x3 with a much smaller window:

AngleCost_2x1 = $\sum_{x=0}^{1} abs(n(x+xoffset,y+1)-n(x-xoffset,y-1))$

AngleCost_2x1 can be collected during the calculation of AngleCost_6x3.

Horizontal Median

One final step is used to prevent sudden angle changes – the angle detected for the pixel pair is compared to the angle detected for the pixels to the right and left and the median of the 3 is the angle finally used:

angle_final(x) = median3(angle(x-2), angle(x), angle(x+2))

2.8.4.3 Spatial Deinterlacer Interpolation

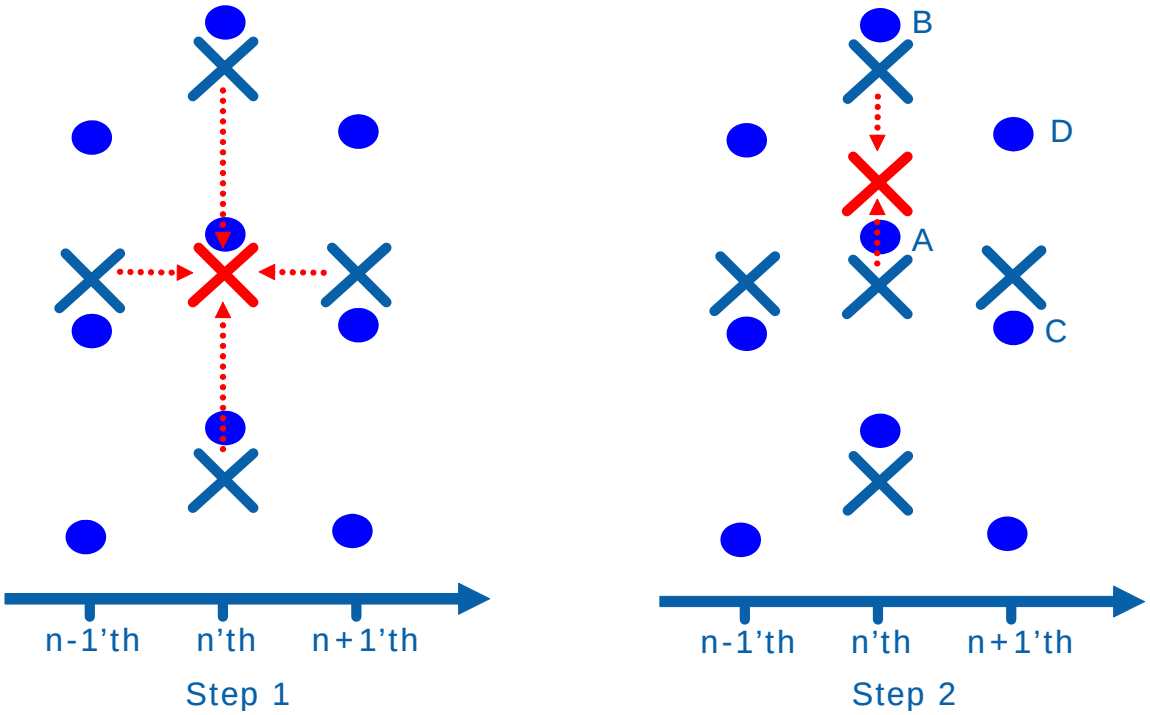
Once the best angle is picked, the interpolation is done on a per pixel basis. Both the chroma and luma need to be interpolated (see section 2.8.4.4 for chroma). Only 422 output is needed, so there will be a chroma pair for each 2 lumas. The interpolation itself is very simple: take a pixel from the line above and the line below along one of the 9 possible angles, and average the 8-bit luma and chroma values to get the result pixel. We will do 2 lumas per clock to get enough performance.

2.8.4.4 Chroma Up-Sampler

The DN/DI block supports 4:2:0, 4:1:1 and 4:2:2 inputs, but only outputs 4:2:2. For 4:2:0 and 4:1:1 the chroma needs to be up-sampled to 4:2:2 before interpolation.

The 4:2:0 input has chroma at ¼ the rate of the luma; ½ in the horizontal and ½ in the vertical directions. The output needs to be 4:2:2, where chroma is ½ the rate of luma; ½ the horizontal but the same in the vertical direction. Then chroma can be de-interlaced in the vertical direction. For luma we are working with 16x4 blocks, so for chroma we will have 8x2 in 4:2:0 and 8x4 in 4:2:2.

The 4:2:0 to 4:2:2 conversion requires doubling the chroma in the vertical direction to match the luma:



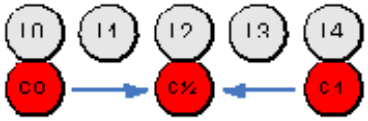
The chroma is doubled by a simple interpolation in both time and space. In the following equations, pixel locations are specified as u(field, x_location, y_location). Field=n would be from the current field, n-1 is from the previous field, and n+1 is from the next field. The Cr and Cb X and Y values are ½ the luma values to map to the smaller area.

```
temporal_cr = (cr(n-1,x,y) + cr(n+1,x,y)) / 2    // Simple average in time


spatial_cr = (cr(n,x,y-1) + cr(n,x,y+1)) / 2       // Simple average in vertical space


if (STMM3 < stmm_min)

   new_cr = temporal_cr

else if (STMM > stmm_max)

   new_cr = spatial_cr

else

  new_cr = ((STMM3 – stmm_min) * spatial_cr  + (stmm_max - STMM3) * temporal_cr) >> stmm_shift
```

Note that this simple chroma interpolation is not correct, since the chroma sample position is ¼ of a pixel different between 420 and 422. The polyphase filter in the scaler will be used to correct this inprecision by modifying the filter coefficients in software.

For performance a single Cr and Cb has to be produce per clock in this stage to match the 2 pixel per clock performance goal.

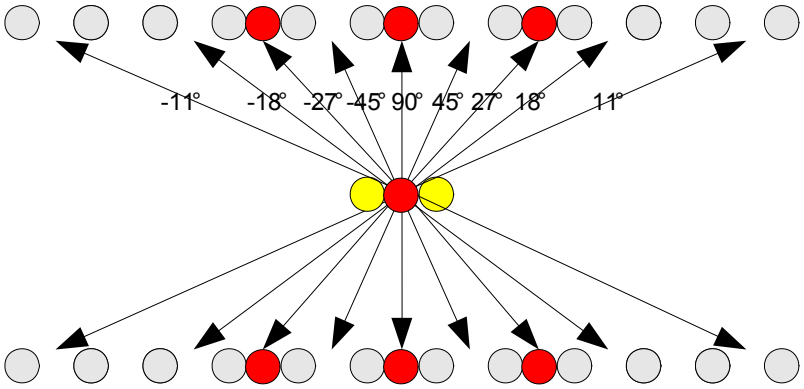
4:1:1 also has chroma at ¼ the rate of luma; ¼ in the horizontal direction and the same in the vertical direction. To convert to 4:2:2 we need to double the chroma horizontally. This will be done by averaging the chromas to the right and left to produce the new chroma.



The above diagram shows how the existing chroma values (both U and V) are averaged between C0 and C1 to produce the new C½. C0 is the chroma asociated with lumas L0 through L3, while C1 is associated with L4 through L7.

2.8.4.5 Chroma Deinterlace

The next step is to do the deinterlacing. Chroma uses the output of the luma angle decision, but reduces the number of angles. The actual spatial deinterlace algorithm is a little different for chroma, since there are only 1 chroma per 2 lumas: some of the chromas are missing and must be filled in.



The diagram shows the chromas used in red. Only 90°, -27° and 27° are directly available. The chromas for +/-45° are derived by a simple average of the 90° and 27° chromas. +/-18° and +/-11° both use the chroma for +/-27°.

2.8.4.5.1 Static Image Fallback Mode

This algorithm has a problem with static images – alternate fields use different luma angle detections and can select different angles, causing noticable flicker. Rather than calculating a separate set of angles for chroma, we instead will blend with STMM so that a static image will use 90 degrees.

```
if (STMM3 < stmm_min)

   chroma_sdi = chroma90degree

else if (STMM > stmm_max)
```

```
    chroma_sdi = chroma_3angle

else

    chroma_sdi = (chroma90degree * (stmm_max – STMM3) + chroma_3angle * (STMM3 – stmm_min))
            >> stmm_shift
```

2.8.4.6 Temporal Deinterlacer and Final Deinterlacer Blend

The temporal deinterlacer is a simple average between the previous and next field; when deinterlacing the 1st field of current the average will be between the 2nd field of previous and the 2nd field of current.

The interpolation between spatial and temporal:

```
if (STMM3 < stmm_min)

   deinterlace_out  = tdi;

else if (STMM3 > stmm_max)

   deinterlace_out = sdi;

else

  deinterlace_out = (sdi * (STMM3 – stmm_min)  + tdi * (stmm_max – STMM3)) >> stmm_shift
```

2.8.4.7 Progressive Cadence Reconstruction

When the FMD for the previous frame indicates that a progressive mode is being used rather than interlaced, the luma and chroma will be taken from adjacent fields rather than spatially interpolated. The exact fields needed depend on state variables written to memory by a thread at the end of the previous frame. The thread will use the FMD variances written to memory via CSunit on the flush at the end of a frame.

Since we are deinterlacing 2 fields at a time – one from the previous frame and one from the current frame (see section 2.8.6.1) we will need a state variable which says how each one should be put together. In each case there are only two possibilities – either the field should be put together with the matching field in the same frame or it should be put together with the adjacent field in the other frame.

If we are deinterlacing the 2nd field from frame N and the 1st field from frame N+1, then the FMD decision (which is made on frame boundaries) will be from frame N-1.

Chroma is reconstructed the same as luma – only the first step of doubling chroma is done in the chroma upsampling block for the two needed fields.

2.8.4.8 Motion Search

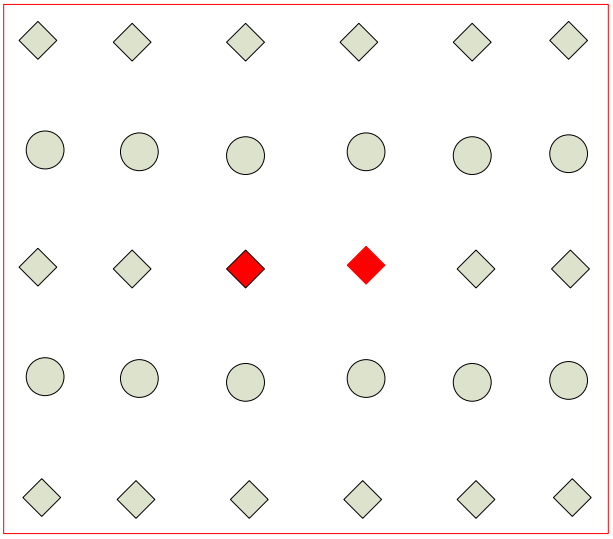
Motion will be estimated independently for each horizontal pair of pixels in the 16x4 block. The area around each pixel pair will be compared to areas in adjacent fields with different X/Y offsets. 16 different offsets, or motion vectors, will be examined in this order:

Y= -2, X = -1, 0, 1

Y = 0, X = -6, -5, -4, -3, -2, 2, 3, 4, 5, 6

Y = 2, X = -1, 0, 1

The area to be compared around the pixel pair is a 6 wide by 5 high window - 2 pixels on right and left and 2 lines above and below. The lines above and below are from both fields, so a total of 3 lines from the same field and 2 lines from the complement field are compared to lines in 2 fields from an adjacent frame.



The motion estimation equation for a pixel pair is:

$$SAD = \sum_{i=x-w}^{x+w+1} \sum_{j=y-h}^{y+h} \left| p_{ref}\left(i + M_x, j + M_y\right) - p_{curr}(i, j) \right|$$

$$\left(h = 2 \; and \; w = 2\right)$$

Mx, My is the motion vector offset being tested, and x,y is the location of the leftmost pixel of the pair. The motion vector with the smallest SAD is kept as the best motion estimate; if two motion vectors have the same SAD then the last one tested will be kept.

2.8.4.9 Robustness Checks

The motion estimate output goes through 2 checks to make sure it is not an aberration – a smoothness check and a consistency check.

2.8.4.9.1 Consistency Check

The consistency check is done per pixel and makes sure that the pixels we are interpolating for MC have a lower delta than the ones that would be interpolated for spatial DI:

$$\left|P_{cur_opp}(x-Edge, y-1)-P_{cur_opp}(x+Edge, y+1)\right| > \left|P_{DI}(x, y)-P_{DI_cur}(x, y)\right|$$
$$\&\&\left|P_{DI}(x, y)-P_{DI_cur}(x, y)\right| < MC_pixel_consistency_TH(default:25)$$

Here Edge is the delta found by SDI which corresponds to the best angle. ***MC_pixel_consistency_TH*** (U6)is a state parameter.

P_{DI_cur} is defined as: (same definition as in the motion compensation section)

• If(Mx%2 == 0 && (My/2)%2 == 0)

$$P_{DI_cur}(x, y)=P_{cur_same}(x-M_x/2, y-M_y/2);$$

• If(Mx%2 == 1 && (My/2)%2 == 0)

$$P_{DI_cur}(x, y)=\begin{cases}\mathrm{AVG}\left(P_{cur_same}(x-M_x/2, y-M_y/2), P_{cur_same}(x-M_x/2-1, y-M_y/2)\right); & \text{if } (M_x \geq 0)\\ \mathrm{AVG}\left(P_{cur_same}(x-M_x/2, y-M_y/2), P_{cur_same}(x-M_x/2+1, y-M_y/2)\right); & \text{if } (M_x < 0)\end{cases}$$

• If(Mx%2 == 0 && (My/2)%2 == 1)

$$P_{DI_cur}(x, y)=\mathrm{AVG}\left(P_{cur_same}(x-M_x/2, y-M_y/2-1), P_{cur_same}(x-M_x/2, y-M_y/2+1)\right);$$

If(Mx%2 == 1 && (My/2)%2 == 1)

$$P_{DI_cur}(x, y)=\begin{cases}\mathrm{AVG}\begin{pmatrix}P_{cur_same}(x-M_x/2, y-M_y/2-1), P_{cur_same}(x-M_x/2-1, y-M_y/2-1),\\ P_{cur_same}(x-M_x/2, y-M_y/2+1), P_{cur_same}(x-M_x/2-1, y-M_y/2+1)\end{pmatrix}; if\,(M_x \geq 0)\\ \mathrm{AVG}\begin{pmatrix}P_{cur_same}(x-M_x/2, y-M_y/2-1), P_{cur_same}(x-M_x/2+1, y-M_y/2-1),\\ P_{cur_same}(x-M_x/2, y-M_y/2+1), P_{cur_same}(x-M_x/2+1, y-M_y/2+1)\end{pmatrix}; if\,(M_x < 0)\end{cases}$$

2.8.4.9.2 Smoothness Check

The smoothness check compares the motion vector found for neighboring pixel pairs. The neighbors are different for different locations to make sure it stays within the local 4x4. Each pixel pair has 3 sets of comparison with neighbor pixel pair within the 4 by 4: 2 sets of X/Y comparisons for the vertical direction and one set of X/Y comparisons for the horizontal direction.

For lines 1 and 2 in the 16x4:

$$If\left(abs\left(MV_x(x, y)+MV_x(x, y+1)\right)\right) <= smooth_mv_th$$
$$AND\ abs\left(MV_y(x, y)+MV_y(x, y+1)\right) <= smooth_mv_th$$
$$AND\left(abs\left(MV_x(x, y)-MV_x(x, y+2)\right)\right) <= smooth_mv_th$$
$$AND\ abs\left(MV_y(x, y)-MV_y(x, y+2)\right) <= smooth_mv_th$$

Where ***smooth_mv_th***(U2) is a state parameter.

This equation ensures that the pixel pair 1 and 2 lines below have motion vector X and Y components (MVx & MVy) that are within a threshold of the best motion vector for the current pixel pair. The compares with y+1 use "+" rather than "-" since they are comparing motion vectors in the opposite field, which have motion vectors pointing in the opposite direction, since they are using the current field as their reference. For example, if the current pixel has a motion vector of (4,2), the motion vector of x,y+1 would be the same if it is (-4,-2).

For lines 3 and 4 in the 16x4:

$$
\begin{aligned}
&If\left(abs\left(MV_x(x,y)+MV_x(x,y-1)\right)\right) <= smooth_mv_th \\
&AND\ abs\left(MV_y(x,y)+MV_y(x,y-1)\right) <= smooth_mv_th \\
&AND\left(abs\left(MV_x(x,y)-MV_x(x,y-2)\right)\right) <= smooth_mv_th \\
&AND\ abs\left(MV_y(x,y)-MV_y(x,y-2)\right) <= smooth_mv_th
\end{aligned}
$$

For pixel pairs with the first pixel location x%4 == 0 (low X in the 4x4):

$$
\begin{aligned}
&If\left(abs\left(MV_x(x,y)-MV_x(x+2,y)\right)\right) <= smooth_mv_th \\
&AND\ abs\left(MV_y(x,y)-MV_y(x+2,y)\right) <= smooth_mv_th
\end{aligned}
$$

For pixel pairs with the first pixel location x%4 != 0 (high X in 4x4):

$$
\begin{aligned}
&If\left(abs\left(MV_x(x,y)-MV_x(x-2,y)\right)\right) <= smooth_mv_th \\
&AND\ abs\left(MV_y(x,y)-MV_y(x-2,y)\right) <= smooth_mv_th
\end{aligned}
$$

When all 3 comparisons pass the threshold, the smoothness check is passed.

2.8.4.10 Motion Comp

The MCDI output is an average done per pixel on pixels chosen from adjacent field.

There are 4 different equations depending on the motion vector (Mx, My):

If (Mx%2 ==0) && (My == 0) then $P_{DI}(x,y) = P_{ref_same}\left(x+M_x/2,\ y+M_y/2\right);$

If (Mx%2 ==1) && (My == 0) then

$$
P_{DI}(x,y) = \begin{cases} \mathrm{AVG}\left(P_{ref_same}\left(x+M_x/2,\ y+M_y/2\right), P_{ref_same}\left(x+M_x/2+1,\ y+M_y/2\right)\right); & \text{if } (M_x \geq 0) \\ \mathrm{AVG}\left(P_{ref_same}\left(x+M_x/2,\ y+M_y/2\right), P_{ref_same}\left(x+M_x/2-1,\ y+M_y/2\right)\right); & \text{if } (M_x < 0) \end{cases}
$$

If (Mx%2==0) && abs(My) == 2 then

$$P_{DI}(x,y) = \mathrm{AVG}\left(P_{ref_same}\left(x + M_x/2,\ y + M_y/2 - 1\right), P_{ref_same}\left(x + M_x/2,\ y + M_y/2 + 1\right)\right);$$

If (Mx%2==1) & abs(My) == 2 then

$$P_{DI}(x,y) = \begin{cases} \mathrm{AVG}\begin{pmatrix} P_{ref_same}\left(x + M_x/2,\ y + M_y/2 - 1\right), P_{ref_same}\left(x + M_x/2 + 1,\ y + M_y/2 - 1\right), \\ P_{ref_same}\left(x + M_x/2,\ y + M_y/2 + 1\right), P_{ref_same}\left(x + M_x/2 + 1,\ y + M_y/2 + 1\right) \end{pmatrix}; if\,(M_x \geq 0) \\ \mathrm{AVG}\begin{pmatrix} P_{ref_same}\left(x + M_x/2,\ y + M_y/2 - 1\right), P_{ref_same}\left(x + M_x/2 - 1,\ y + M_y/2 - 1\right), \\ P_{ref_same}\left(x + M_x/2,\ y + M_y/2 + 1\right), P_{ref_same}\left(x + M_x/2 - 1,\ y + M_y/2 + 1\right) \end{pmatrix}; if\,(M_x < 0) \end{cases}$$

For all these equations, if more vareties of My are used than -2,0,2 then we need to use (My/2)%2==0) instead of My==0, and (My/2)%2==1 instead of abs(My)==2.

2.8.4.11 Merge with TDI & SDI

The MADI equation was:

if (STMM3 < stmm_min)

deinterlace_out = tdi;

else if (STMM3 > stmm_max)

deinterlace_out = sdi;

Else

deinterlace_out = ((STMM3 – stmm_min) * sdi + (stmm_max - STMM3) * tdi) >> stmm_shift

Where STMM3 is a measure of the complexity of the scene and how much motion is in it.

The equation with MCDI is:

if (STMM3 < stmm_min)

    Deinterlace_out = tdi;

else if (STMM3 > stmm_max)

    deinterlace_out = DItemp;

else

    deinterlace_out = ((STMM3 – stmm_min) * DItemp + (stmm_max - STMM3) * tdi) >> stmm_shift

Where DItemp is defined below:

Content Adaptive Thresholding:

We denote the best_ME_SAD as the minimal SAD value for the MV candidates. Best_ME_SAD and Best_SAD_Angle_cost are measured based on the block of pixels. The new control equation with MCDI is calculated per pixel:

If ((best_ME_SAD <= ***CAT_TH1***)

If (Consistency check is passed && Smoothness check is passed)

DItemp = MCDI;

Else

DItemp = sdi;

Else if (***CAT_TH1***<best_ME_SAD < CAT_TH2****30***) {

If (Consistency check is passed && Smoothness check is passed) AND

(SDI_angle =90 degree) AND

(best_ME_SAD + ***SAD_Tight_TH*30*** < Best_SAD_Angle_cost*2) AND

{(MCDI==median3(MCDI, $P_{curr_pixel}(x, y-1)$, $P_{curr_pixel}(x, y+1)$) ||

(Min[abs(MCDI - $P_{curr_pixel}(x, y-1)$), abs(MCDI - $P_{curr_pixel}(x, y+1)$)] <

NeighborPixel_TH)}

DItemp = MCDI;

Else

DItemp = sdi;

} Else

DItemp = sdi

Where ***CAT_TH1***(U2, default = 0), ***SAD_Tight_TH*** (U4, default=5) and ***NeighborPixel_TH***(U4, default=10) are state parameters. CAT_TH2 is a content adaptive value dependent on SCM. SCM = SHCM+SVCM from the spatial complexity measurement.

If (SCM < SCM_A)

CAT_TH2 = ***SAD_THA***;

Else if (SCM > SCM_B)

CAT_TH2 = ***SAD_THB***;

Else

CAT_TH2 = SCM / ***CAT_slope;***

Where ***CAT_slope*** (U4: default value 10). ***SAD_THA*** (U4, default 5) and ***SAD_THB*** (U4, default 10) are state parameters, and SCM_A and SCM_B are derived parameters:

SCM_A = ***CAT_slope * SAD_THA***; // 4-bit * 4-bit to produce 8-bit value

SCM_B = ***CAT_slope * SAD_THB***; // 4-bit * 4-bit to produce 8-bit value

2.8.5 Field Motion Detector

The Field Motion Detector is generated in either the EU or in the driver with a set of differences gathered across entire fields. It is used to detect when a non-interlaced source like a film has been converted to interlaced video – in this case there will be pairs of fields which can be put back together to make frames rather than interpolating. The variances for the block are sent to the VSCunit to be summed across the entire frame. The results are available in MMIO registers.

2.8.5.1 Simple Differences

The first set of variances are simply a sum of absolute pixel differences. The equations are done for every pixel with an even y coordinate:

variance[0] += Diff_cTpT = (c(x,y) – p(x,y)) ^ 2; – difference between pixels from the top fields of the current and previous frame.

variance[1] += Diff_cBpB = (c(x,y+1) – p(x,y+1)) ^ 2; – difference between pixels from the bottom fields of the current and previous frame.

variance[2] += Diff_cTcB = (c(x,y) – c(x,y+1)) ^ 2; – difference between pixels from the top field and bottom field in the current frame.

variance[3] += Diff_cTpB = (c(x,y) – p(x,y+1)) ^ 2; – difference between pixels from the top field of the current frame and bottom field of previous frame.

variance[4] += Diff_cBpT = (c(x,y+1) – p(x,y)) ^ 2; – difference between pixels from the bottom field of the current frame and top field of previous frame.

The variances summed for each 16x4 block are divided by 16 before adding them to the sum for the frame to make sure the frame-level sum fits in a 32-bit register.

2.8.5.2 Counter Variances

The rest of the variances are counters for variance conditions as described in the following code:

```
// Same field difference of the current frame

diff_cTcT = (c(x,y) – c(x,y+2)) ^ 2;

diff_cBcB = (c(x,y-1) – c(x,y+1)) ^ 2;

// Same field difference of the previous frame

diff_pTpT = (p(x,y) – p(x,y+2)) ^ 2;
```

```
diff_pBpB = (p(x,y-1) – p(x,y+1)) ^ 2;
// Same field vertical smoothness of the current frame
diff_cT = ABS(c(x,y) – c(x,y-2)) + ABS(c(x,y) – c(x,y+2)) –  ABS(c(x,y-2) + c(x,y+2));
diff_cB = ABS( c(x,y+1) – c(x,y-1) ) + ABS( c(x,y+1) – c(x,y+3) ) –
ABS( c(x,y-1) + c(x,y+3) );
if( diff_cTpT + diff_cBpB > fmd_tdiff ) {                    // if moving pixels,

   // Fine tears for cadence detection except 2-2 detection
   if( diff_cTcB > diff_cTcT + diff_cBcB)                    variance[5]++;
   else                                                              variance[6]++;

   // Find tears for 2-2 cadence detection
   if( diff_cT < fmd_vdiff1 && diff_cB < fmd_vdiff1) {     // if fields are vertically smooth,

variance[7]++;    // total moving pixels

// Find tears. (1st condition is to exclude very small variations)
if(diff_cTcB >=fmd_vdiff2 && diff_cTcB > diff_cTcT + diff_cBcB)          TEAR_1(x,y) = 1
if(diff_cTpB >=fmd_vdiff2 && diff_cTpB > diff_cTcT + diff_pBpB)          TEAR_2(x,y) = 1
if(diff_cBpT>=fmd_vdiff2 && diff_cBpT > diff_pTpT + diff_cBcB)           TEAR_3(x,y) = 1
   }
}
```

2.8.5.3 Tear Variances

The all 3 TEAR_N variables are compared to neighbors to eliminate strays:

```
if(TEAR_N(x-1,y) == 0 &&
  TEAR_N(x+1,y) == 0 &&
  TEAR_N(x,y-2) == 0 &&
  TEAR_N(x,y+2) == 0)                TEAR_N(x,y) = 0; where N=1,2,3.
variance[8] = sum of TEAR_1(x,y)
```

variance[9] = sum of TEAR_2(x,y)

variance[10] = sum of TEAR_3(x,y)

if (variance[8] > variance[9] && variance[8] > variance[10])

variance[7] = variance[8] = variance[9] = variance[10] = 0

if (variance[8] < fmd_thr_tear) variance[8] = 0

if (variance[9] < fmd_thr_tear) variance[9] = 0

if (variance[10] < fmd_thr_tear) variance[10] = 0

The variances are summed for each block across the frame. The accumulators may require 24-bit adders if the differences are 8-bits and there can be 128 (horizontally) * 256 (vertically) of them. The sums are written to memory at the end of the frame.

Two sets of FMD variances are needed to support 2 simultaneous streams. The streams are distinguished by the dndi_stream_id state variable in the DI state.

[DevILK] A-Stepping Erratum: TEAR_N compute doesn't follow the equation above. Two signals were missing, thus, it is incorrectly calculated as the following. Without the added protection of the N=-2 & N=4 collection of feature, the robustness of 2:2 detection suffers.

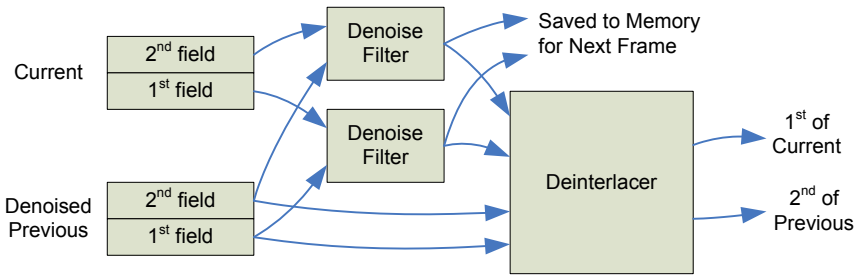
if(TEAR_N(x-1,y) == 0 &&

TEAR_N(x+1,y) == 0 &&) *TEAR_N(x,y) = 0; where N=1,2,3.*

2.8.6 Implementation Overview

2.8.6.1 Input and Output Frames

Two frames are needed to do deinterlacing, but for any two frames, two fields can be deinterlaced, doubling the output for the same input bandwidth. This also allows the denoise filter to only filter a frame once.



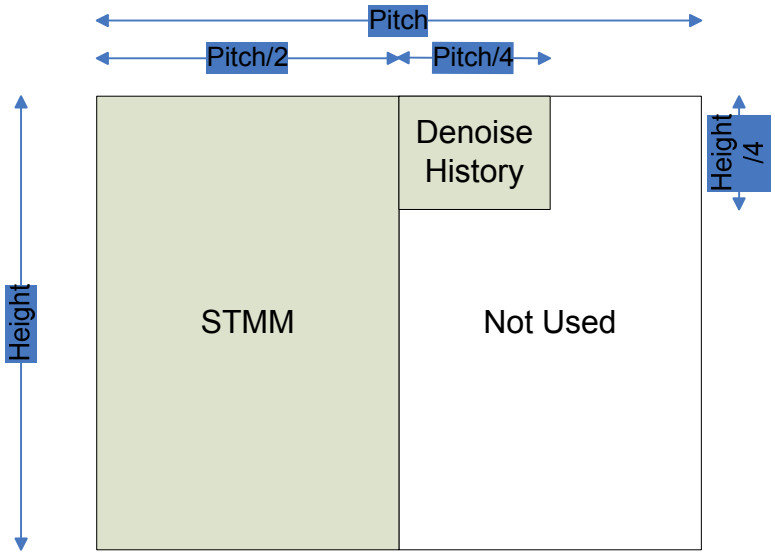
The above picture shows that two frames are read in, called current and previous. The two fields of the next frame are denoised using adjacent fields. The 2nd field of previous can be deinterlaced using current as the reference, and the 1st field of current can be deinterlaced using previous as reference.

Since we are producing 2 16x4 outputs, and the performance goal is to output 2 pixels per clock, we have 64 clocks to run 2 denoise filters and 2 deinterlacers.

The fields are referred to as 1st and 2nd because either the top or bottom field can be the first in the sequence depending on a state variable.

2.8.6.1.1 Statistics Surface Memory Format

The statistics memory page is used to store both STMM and Denoise history. The STMM and Denoise history are stored in separate areas addressed by a single base address pointer:



The STMM for any pixel pair is addressed by:

STMM_X = pixelX / 2

STMM_Y = pixelY

The Denoise History for any 4x4 block is addressed by

DH_X = Pitch/2 + pixelX/4

DH_Y = pixelY/4

Where the pixelX/Y comes from the address of the left pixel for STMM and the upper-left pixel for the Denoise History. The Pitch is from the surface state.

The read and write surfaces for each frame must be separate, since any individual block will not know if the neighbor blocks have been updated yet. This can be implemented as a ping-pong buffer pair with the write surface for each frame becoming the read surface for the next.

2.8.6.2 First Frame Special Case

The first frame in the sequence is a special case for both denoise and deinterlace. Only data from the current frame address is read, the previous frame, clean previous, statistics and control addresses are ignored. Behavior for each function is as follows:

1) Denoise – The denoise filter needs to use the spatial filter, since there is no previous frame from which to do a temporal filter.

    a. The Denoise Motion History is not read.

    b. The blend between the temporal and spatial is forced to 100% spatial.

    c. [DevSNB+]The Denoise Motion History output values are written to 0.

2) BNE – The Block Noise Estimate only uses current frame values and so works normally.

3) Deinterlacer – Only the 1st field of the current frame frame is deinterlaced in this case – the 2nd of previous does not exist.

    a. The spatial deinterlacer is used to produce the output.

    b. The STMM input values are not read.

    c. The STMM output values are written as a the maximum 255 value so that the next frame is correctly told that spatial deinterlacing was used in this frame.

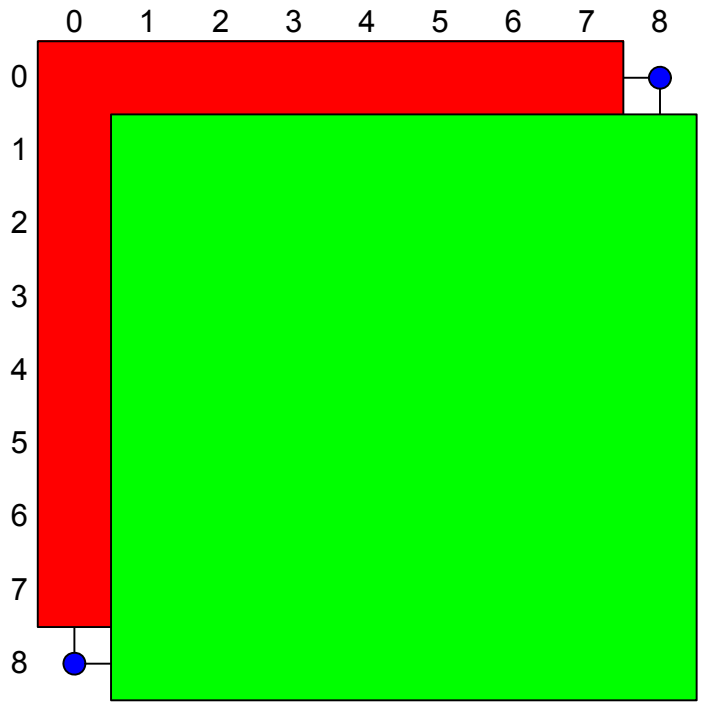
4) FMD – variances between the top and bottom of the current field should be output correctly. Variances that read from the previous field should indicate a maximum difference.

5) Progressive Cadence Reconstruction – the FMD input is not read, so always assume interlaced.

2.9 Adaptive Video Scaler

The adaptive video scaler consists of a pair of filters. The sharp filter is an 8x8 and the smooth filter is bilinear. The results of the two filters are alpha blended together using an alpha factor determined separately from an algorithm that examines the pixel values in the each vector.

There are a total of four different coefficient tables with two in each direction. For both directions is it possible to use either of the two tables that are assigned to it or use both at once with one table for the Y and the other table for the U/V. The coefficients are programmable by software and loaded via a new command streamer instruction. The coefficients are considered to be nonpipelined state, with a full pipeline flush being required before a new set of coefficients is loaded.



The above diagram shows two pixels (red and green) mapped onto a texture map, with the texel centers blue. The red/green boxes around the pixels indicate the area where the pixel would choose the same 8x8 footprint for its filter, while the large transparent box indicates the footprint for each pixel.

The u/v addresses for each pixel (in texel space) are as follows:

red pixel: u=3.3, v=3.3 (betau=0.3, betav=0.3)

green pixel: u=4.3, v=4.7 (betau=0.3, betav=0.7)

The integer u/v address of the upper left pixel of the footprint is a function of the pixel u/v address as follows:

u(UL) = floor(u(pix)) – 3

v(UL) = floor(v(pix)) – 3

When the 8x8 filter is selected, the 8x8 texel block surrounding the pixel sample point is selected. The blend factors "beta" (horizontal and vertical) are determined by the relative distance between the pixel center and the nearest 4 texels (2x2). The betas are first truncated to 5 bits (*i*).

The beta value is used to look up two sets of 8 coefficients, one set of 8 for horizontal (called K_h0..7), and one set of 8 for vertical (called K_v0..7).

2.9.1 Filtering Operations

There are two separate filters, sharp and smooth, which are blended in an adaptive manner.

2.9.1.1 Sharp

The following formula is used to compute the filtered texture color for the sharp filter:

$R0 = T00*K_h0 + T01*K_h1 + T02*K_h2 + T03*K_h3 + T04*K_h4 + T05*K_h5 + T06*K_h6 + T07*K_h7$

$R1 = T10*K_h0 + T11*K_h1 + T12*K_h2 + T13*K_h3 + T14*K_h4 + T15*K_h5 + T16*K_h6 + T17*K_h7$

$R2 = T20*K_h0 + T21*K_h1 + T22*K_h2 + T23*K_h3 + T24*K_h4 + T25*K_h5 + T26*K_h6 + T27*K_h7$

$R3 = T30*K_h0 + T31*K_h1 + T32*K_h2 + T33*K_h3 + T34*K_h4 + T35*K_h5 + T36*K_h6 + T37*K_h7$

$R4 = T40*K_h0 + T41*K_h1 + T42*K_h2 + T43*K_h3 + T44*K_h4 + T45*K_h5 + T46*K_h6 + T47*K_h7$

$R5 = T50*K_h0 + T51*K_h1 + T52*K_h2 + T53*K_h3 + T54*K_h4 + T55*K_h5 + T56*K_h6 + T57*K_h7$

$R6 = T60*K_h0 + T61*K_h1 + T62*K_h2 + T63*K_h3 + T64*K_h4 + T65*K_h5 + T66*K_h6 + T67*K_h7$

$R7 = T70*K_h0 + T71*K_h1 + T72*K_h2 + T73*K_h3 + T74*K_h4 + T75*K_h5 + T76*K_h6 + T77*K_h7$

$F' = R0*K_v0 + R1*K_v1 + R2*K_v2 + R3*K_v3 + R4*K_v4 + R5*K_v5 + R6*K_v6 + R7*K_v7$

F_sharp = Clamp F' to [0.0, 1.0)

where:

- T*rc* is the texel color in row *r* ([0..3]) and column *c* ([0..3]) of the 8x8 array of neighboring texel colors
- F_sharp is the final output color of the sharp filter.

2.9.1.2 Smooth

The following formula is used to compute the filtered texture color for the smooth filter:

F_smooth = (T33 * (1-betaU) + T34 * betaU) * (1-betav) + (T43 * (1-betaU) + T44 * betaU) * betav

2.9.1.3 Adaptive Filtering

The adaptive filter only supports RGB or YUV packed formats. For YUV formats, the alpha value is determined only by the Y channel (green), with this alpha value being applied to all three channels. For the RGB formats the alpha value is determined based on an average of all three channels with G having double the weight as the other channels.

Each horizontal or vertical filter has 8 texels input which feeds into an eight tap filter. On the center two there is a linear blend using the betaV. Then using the Y channel an adaptive part weight is calculated and the two filters are alpha blended. The adaptive part calculated on the Y channel is used on all three channels. Only the 8 MSBs are used in these calculations.

The adaptive part is done to classify a pixel as prone to ringing or not. This is done by analyzing the 8 Y samples from the interpolation window ($Wy_0 \ldots Wy_7$).

When the pixels are in an RGB format, Y is extracted from the RGB components in window W:

$Wy_i = (Wr_i + 2*Wg_i + Wb_i)/4; \quad 0<=i<=7$

There are 3 measurements on these samples that decide how to act. The result is a number between zero and one.

Analysis is performed on Y samples in 8 bit precision.

Measurement #1 –1st derivatives on center samples (minimum of 2 maximums).

maxDeriv4_a = max(|Wy3-Wy4|, |Wy2-Wy3|)

maxDeriv4_b = max(|Wy3-Wy4|, |Wy4-Wy5|)

maxDeriv4 = min(maxDeriv4_a, maxDeriv4_b)

Measurement #2 –2nd derivatives on center samples (minimum of 2 maximums).

Deriv1 = Wy2-Wy3; Derive2 = Wy3-Wy4; Deriv3 = Wy4-Wy5

Deriv2a = |Deriv1-Deriv2|

Deriv2b = |Deriv3-Deriv2|

Deriv2Avg = (Deriv2a + Deriv2b)/2

D4 = min(Deriv2Avg, maxDeriv4)

Measurement #3 – 1st derivative on all (8) Y samples.

$maxDeriv8 = max(|Wy_m - Wy_{m+1}|); \; 0 \le m \le 6;$

When *D4* is small enough and *maxDeriv8* is large enough then ringing can appear. So 2 alphas are calculated (one for *D4* and one for *maxDeriv8*), and the minimum of the two is used as the sharpness alpha. An alpha of 255 means the Polyphase scaler is used and an alpha of 0 means that the linear scaler is used.

$$D4Alpha = \begin{cases} D4 \le \mathbf{MaxDerivPoint4} & 0 \\ D4 \ge \mathbf{MaxDerivPoint4} + 2^{8-\mathbf{MaxDeriv4SlpBits}} & 255 \\ else & \left(D4 - \mathbf{MaxDerivPoint4}\right) \cdot 2^{8-\mathbf{MaxDeriv4SlpBits}} \end{cases}$$

$$D8Alpha = \begin{cases} maxDeriv8 \le \mathbf{MaxDerivPoint8} & 255 \\ maxDeriv8 \ge \mathbf{MaxDerivPoint8} + 2^{8-\mathbf{MaxDeriv8SlpBits}} & 0 \\ else & 255 - \left(\left(maxDeriv8 - \mathbf{MaxDerivPoint8} \right) \cdot 2^{8-\mathbf{MaxDeriv8SlpBits}} \right) \end{cases}$$

Note that multiplying by an exponent of 2 is implemented as bit shifts.

Calculate *SharpnessAlpha* (U0.8 precision):

SharpnessAlpha=max*(D8alpha, D4Alpha)*

if ((xDirection ? xAdaptiveBypass : YAdaptiveBypass) == 1) Then (*SharpenessAlpha* = *SharpnessLevel*)

$$Y_{out} = \begin{cases} SharpnessAlpha = 255 & Y_1 \\ else & Y_2 + \dfrac{128 + (Y_1 - Y_2) \cdot SharpnessAlpha}{256} \end{cases}$$

The UV results are handled in the same manner.

2.10 Image Enhancement Filter and Video Signal Analysis

The IEF module takes in the YUV 444 color space with 10 bit components.

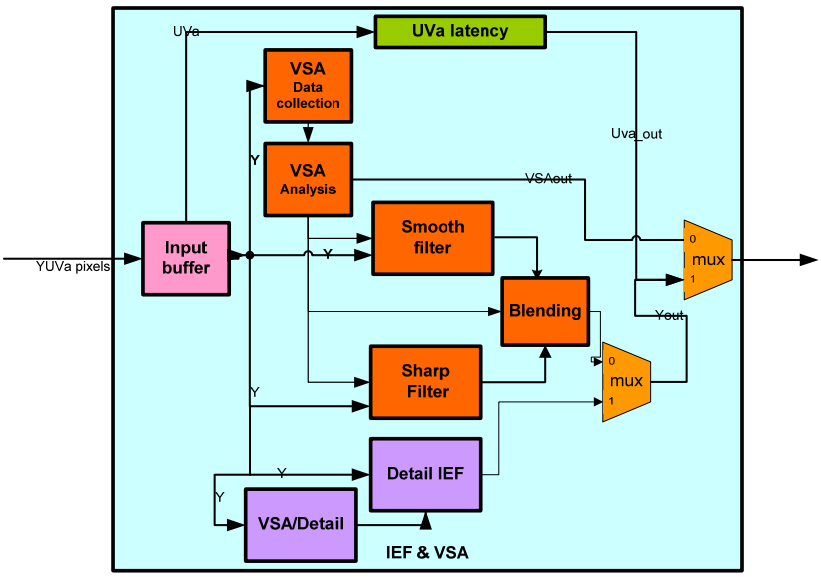
The IEF and VSA have 3 optional modes of operation: basic detail filter 3x3 mode, basic detail filter 5x5 mode and the combination mode. Detail Filter 3x3 mode which is a simple Sobel as VSA and 9 tap constant IEF. Detail Filter 5x5 mode which is a simple Sobel as VSA and 9 tap constant IEF on a sparse 5x5 environment. The combination mode is the full VSA mode and 25 tap filtering doing sharpening and/or smoothing. Either the detail filter mode or combination mode can be removed at synthesis.

VSA – Video Signal Analysis – analyzes the local Y environment of each pixel and outputs several values that describe its nature (smooth, detailed, sharpening). Those values will be used by the IEF to decide how the filter should be applied at each pixel location.

IEF – Image Enhancement Filter – The operations this filter performs are detail filter, smoothing and sharpening on the Y component, according to the VSA outputs.

The IEF throughput is 2 pixels per clock.

2.10.1 Block Diagram



2.10.2 Detail Filter Algorithm

2.10.2.1 VSA for Detail Filter

In the VSA for the detail flter mode, Sobel edge detection is used to set different weighting for detail filtering.

$$E_h = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad E_v = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

The edge metric *(EM)* for the target pixel x is formulated as the convolution of the weighting with its 3x3 neighborhood *NH9(x)* as

[DevSNB-Astep]

*EM(x) = |NH9(x) * E_h| + |NH9(x) * E_v| // where the input is 10 bits, EM is 4 bits (*CLIP((|NH9(x) * E_h| + |NH9(x) * E_v|+8) >> 4 , 0 ,15))

[DevSNB-Bstep]

*EM(x) = |NH9(x) * E_h| + |NH9(x) * E_v| // where the input is 10 bits, EM is 4 bits (*CLIP((|NH9(x) * E_h| + |NH9(x) * E_v|+4) >> 3 , 0 ,15))

If (EM(x) > **Strong_Edge_Threshold**) local_adjust = **Strong_Edge_Weight** // local_adjust is 3bits

Else if (EM(x) > **Weak_Edge_Threshold**) local_adjust = **Regular_Weight**

Else local_adjust = **Non_Edge_Weight**

The **Strong_Edge_Threshold, Weak_Edge_Threshold, Strong_Edge_Weight, Non_Edge_Weight** and **Regular_Weight** are the pipelined state variables to be specified by driver. **Strong_Edge_Threshold** & **Weak_Edge_Threshold** are 4-bit length variables.

Min and Max on the 3x3 neighborhood are found and diff3 = Max – Min is calculated. Similarly diff5 represents the difference calculated based on 5x5 neighborhood.

2.10.2.2 Detail IEF

In the mode of detail filter 3x3, the below 2-Dimensional formula is used to extract the high frequency component from the 3x3 neighborhood.

$$sigma(Xc)(2nd_gradient) = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

With the current pixel Xc with the 3x3 neighborhood below, the equation is

X1 X2 X3

X4 Xc X5

X6 X7 X8

Sigma(Xc)(2nd Gradient) = 8 * Xc - (X1+X2+X3+X4+X5+X6+X7+X8) // 13 bits

In the mode of detail filter 5x5, the below 2-Dimensional formula is used to extract the high frequency component from the neighborhood.

$$sigma(Xc)(2nd_gradient) = \begin{bmatrix} -1 & 0 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 8 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 0 & -1 \end{bmatrix}$$

The current pixel is Xc with the 5x5 neighborhood, the equation for 5x5 is

X0 X1 X2 X3 X4

X5 X6 X7 X8 X9

Xa Xb Xc Xd Xe

Xf Xg Xh Xi Xj

Xk Xl Xm Xn Xo

The basic equation is

*Sigma(Xc)(2nd_gradient)=8*Xc – (X0+X2+X4+Xa+Xe+Xk+Xm+Xo)* // 13 bits

The filter used here is the none-directional filter and so different coefficients can be applied to each of the outer 5x5 ring, where the middle pixel is subtracted from each pixel so the sum of the filter's coefficients is 0.

Clipping:

The clipping is utilized to limit the range of the calculated Sigma(Xc) to be among min_clip and max_clip.

min_clip = -1<< (5 + SrcPrecision - 8)

max_clip = (1<< (5 + SrcPrecision - 8)) - 1

(SrcPrecision = 8 for 8-bit video, =10 for 10-bit video)

Thus, min_clip <= clipped(Sigma(Xc)) <= max_clip.

The **Gain_Factor** is the state variable specified by users, local adjust is the result of the VSA, diff3 is max-min in the 3x3 neighborhood. The equation below gives the delta from the original pixel:

Delta(Xc) = (clipped(sigma(Xc)) * gain_factor * local_adjust + 64) / (128*clipped(8+diff3)) (delta is 7 bits, and clipped(8+diff3) is between (0,255)

[DevSNB]

{In HW implementation.

Delta(Xc) = ((clipped(sigma(Xc)) * gain_factor * local_adjust + 64) * (m_DivTable[clipped(8+diff3)]) >> 7)}

2.10.3 Combination mode

2.10.3.1 VSA Analysis

In the VSA for the combo mode, the operation on the 5x5 neighborhood of the Y channel is assumed.

Diff (local contrast) is used as the main criteria. The local contrast result obtained from the diff criteria is fine tuned using global noise measure and other measurements from the VSA. Diff5 and diff3 are

compared, because diff3 measures variability over a smaller region, it is multiplied by 3/2, the larger of the 2 is used as the basic parameter to estimate the smoothness strength. However if sharpness operation is performed the smaller of the 2 is used.

The mapping relation between filtering strength and the estimated variability is modeled using a piece wise linear (PWL) function to linearly interpolate the values among control points. The PWL parameters might vary depending on clip resolution, screen resolution, or other blocks in the video chain such as ACE. Using a PWL enables responding to specific clip features which will be measured by other modules (SW implemented).

8 points are used to divide the mapping range into 7 segments for PWL function. By default the value 0 is used as the Point 0 and the value 255 is used as the Point 7. Points 1 to 6 are specified by driver. Also, Slopes 0 to 6 and Bias 0 to 6 are specified by drivers. There are two sets of Point, Slope and Bias for the case of 3x3 and the case of 5x5. The pseudo code to implement PWL is as followed// (x[i],y[i]) and (x[i+1],y[i+1]) PWL(diff,PNT,BIAS5,SLP5)

```
PWL(diff,Point,Bias,Slope)

        if(Point[end] <= diff) //end =7 in this case

                i = end

        else

                find i  such that Point[i] <= diff <Point[i+1]

return Interpolation = MIN(MAX(((diff – Point[i]) * Slope[i])/8 + Bias[i]),0),255)
```

Gradient analysis

The gradient is defined to be derived based on 2x2 pixels. On a 5x5 neighborhood, there will be 16 (4x4) gredients for the overlapping 2x2 units. dx and dy are calculated using the below convolution masks

For dx

+1	-1
+1	-1

for dy

1	1

-1	-1

norm_grad = (abs(dx) + abs(dy)) is calculated on the 4x4 overlapping window.

And MaxNorm is the largest norm _grad in the 4x4 window.

Measurements of Multi-Ridge & Steepness

MR (multi ridge) is the ratio between the total of all norm_grad in the 4x4 window and the difference between minimum and maximum on the 5x5 window.

$$\text{tot_norm} = \sum_{-2<i<2}^{-2<j<2} \text{norm_grad}(i,j)$$

The total_norm is modified by the difference between minimum and maximum on the 5x5 window.

tot_norm -= 23* (max5 - min5)>>1; // zero if negative

MR = (5*(tot_norm / 8)) / (max5 - min5 +1) // 4 bit division

Dif5_mod = ((3*(max5 - min5))/8) +1

The norm is modified based on Dif5_mod

max_norm_mod= MAX(2*MaxNorm– Dif5_mod)/4,0) // 9.0u

Steepness = max_norm_mod/ Dif5_mod //4.0u. 4 bit division

2.10.3.1.1 Modify diff according to Global Noise Estimatiodenote

The GN1 is denoted as the Global Noise Estimation derived by software driver. The diff is modified based on the GN1 and the pixel intensity

modify_diff5 = diff5 – GN1

modify_diff3 = diff3 - (GN1 > 0? GN1: GN1/2),0))

diff = MAX(MIN(MAX(modify_diff5 , modify_diff3 + (modify_diff3)/2), 1),255) // 8.0u

```
if(diff > Pwl1_pnt3)

   diff = MIN(modify_diff5 , modify_diff3 + (modify_diff3 >> 1)
```

2.10.3.1.2 The Weightings of Sharpening and smoothing strength

The weightings of sharpening and smoothening filter is based on the PWL conditioned on the modified diff.

Sharpening_strength = PWL(diff,PNT,BIAS5,SLP5) // 8.0u

Smoothing_strength = PWL(diff,PNT,BIAS3,SLP3) // 8.0u

And the sharpening weighting is further modified by the measurements of steepness and the multi-grid.

steepness = steepness - MAX(8 -(diff3/2),0); // steepness disabled when diff is very low

Sharpening_strength = Sharpening_strength *(16 – MIN((MR – MR_Threshold)* MR_Boost + (steepness – Steepness_Threshold)* Steepness_Boost),15))/16 // 8.0u

Where MR_Threshold, MR_Boost, Steepness_Threshold and Steepness_Boost are the parameters specified by driver.

2.10.3.2 Sharpening Filtering

R5c	R5cx	R5x	R5cx	R5c
R5cx	R3c	R3x	R3c	R5cx
R5x	R3x		R3x	R5x
R5cx	R3c	R3x	R3c	R5cx
R5c	R5cx	R5x	R5cx	R5c

Non-directional IEF

The location of filter coefficients

The filter of the combinational mode is symmetric.

P(-2,-2)-P(0,0)	P(-1,-2)-P(0,0)	P(0,-2)-P(0,0)	P(1,-2)-P(0,0)	P(2,-2)-P(0,0)
P(-2,-1)-P(0,0)	P(-1,-1)-P(0,0)	P(0,-1)-P(0,0)	P(1,-1)-P(0,0)	P(2,-1)-P(0,0)
P(-2,0)-P(0,0)	P(-1,0)-P(0,0)		P(1,0)-P(0,0)	P(2,0)-P(0,0)
P(-2,1)-P(0,0)	P(-1,1)-P(0,0)	P(0,1)-P(0,0)	P(1,1)-P(0,0)	P(2,1)-P(0,0)
P(-2,2)-P(0,0)	P(-1,2)-P(0,0)	P(0,2)-P(0,0)	P(1,2)-P(0,0)	P(2,2)-P(0,0)

D(i,j) = P(i,j)-P(0,0) as the difference of the target (center) pixel, P(0,0), from the neighboring pixels, P(i,j), shown in the above figure.

Sharp = R5C*(D(2,0) + D(-2,0) + D(0,-2) + D(0,2)) +

R5X*(D(2,2) + D(-2,2) + D(2,-2) + D(-2,-2)) +

R5CX*(D(2,1) + D(-2,1) + D(1,-2) + D(-1,-2) + D(-2,-1) + D(2,-1) + D(1,2) + D(-1,2))

R5C, R5X and R5CX are the paramters specified by driver.

2.10.3.3 Smoothing Filter

Similar to the content adaptive spatial filter in Section 1.8.2.4, smoothing filter is using only neighboring pixels whose value is close to the center pixel value. Global noise is used as a threshold to decide if a pixel value is close to the center pixel. Only pixels whose distance from the center pixel is less than the global noise are used for smoothing.

For each pixel in the 3x3 neighborhood:

```
If(D(i,j) < GN1) D(i,j) = D(i,j)

Else D(i,j) = 0
```

The number of pixels that are not zeroed are counted for the coefficient R3C & R3X individually as NZC and NZX. The factor (NZC, NZX) is then multiplied by each coefficient depending on how many pixels it multiplies. The pseudo code to derive NZC and NZX are as follows.

```
NZX = 0

NZC = 0

For (-2 <= i, j <=2) {

        If (ABS(D(i,j) < GN1) {

                If (i==0 || j==0) NZC ++;

                Else NZX ++;

        }

}
```

Apply smoothing operation

```
Smooth = R3C*(D(1,0) + D(-1,0) + D(0,-1) + D(0,1)) * NZ[NZC] +

          R3X*( D(1,1) + D(-1,1) + D(1,-1) + D(-1,-1)) * NZ[NZX]  // 12.2u round 3 lsb, check for overflow
```

2.10.3.4 Filter Blending

Smoothing filter reduces the power of some or all of the frequencies in the image, while sharpening filter enhance some of the frequencies in the image. The output of filtering is based on the blending of both filterings.

Filtering = -sharp_strength * Sharp + smooth_strength * Smooth // 11.0s round 10bits, check for overflows

Output_pixel = orginal_pixel + filtering // 10.0u

Limiting the Output Pixel

The limiter is applied to constrain the effect of overshoot and undershoot.

If (Output_pixel > max5)

Output_pixel =(Output_pixel - max5) * Maximum_Limiter + max5

Output_pixel = MIN(max5 + Clip_Limiter + ((max5 - max3)*Limiter_Boost), Output_pixel);

else if(Output_pixel < max5)

Output_pixel = min5 - (min5 - Output_pixel)* Minimum_Limiter

Output_pixel = MAX(min5 - (Clip_Limiter + ((min5 - min3) *Limiter_Boost)), Output_pixel)

Maximum_Limiter, Minimum_Limiter, Limiter_Boost and Clip_Limiter are the parameters specified by driver.

2.11 State

2.11.1 BINDING_TABLE_STATE

The binding table binds surfaces to logical resource indices used by shaders and other compute engine kernels. It is stored as an array of up to 256 elements, each of which contains one dword as defined here. The start of each element is spaced one dword apart. The first element of the binding table is aligned to a 32-byte boundary.

DWord	Bit	Description
0	31:5	**Surface State Pointer.** This 32-byte aligned address points to a surface state block. This pointer is relative to the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved : MBZ

2.11.2 SURFACE_STATE

The surface state is stored as individual elements, each with its own pointer in the binding table. Each surface state element is aligned to a 32-byte boundary.

Surface state defines the state needed for the following objects:

- texture maps (1D, 2D, 3D, cube) read by the sampling engine
- buffers read by the sampling engine
- constant buffers read by the data cache via the data port
- render targets read/written by the render cache via the data port
- streamed vertex buffer output written by the render cache via the data port
- media surfaces read from the texture cache or render cache via the data port
- media surfaces written to the render cache via the data port

2.11.2.1 SURFACE_STATE for most messages

2.11.2.1.1 SURFACE_STATE for most messages [DevSNB]

SURFACE_STATE

Project: [DevSNB]

This is the normal surface state used by all messages that use SURFACE_STATE except *deinterlace* and *sample_8x8*.

DWord	Bit	Description
0	31:29	**Surface Type** Project: All Format: U3 enumerated type FormatDesc This field defines the type of the surface.

Value	Name	Description	Project
0h	SURFTYPE_1D	Defines a 1-dimensional map or array of maps	All
1h	SURFTYPE_2D	Defines a 2-dimensional map or array of maps	All
2h	SURFTYPE_3D	Defines a 3-dimensional (volumetric) map	All
3h	SURFTYPE_CUBE	Defines a cube map or array of cube maps	All
4h	SURFTYPE_BUFFER	Defines an element in a buffer	All
5h-6h	Reserved		All

		7h SURFTYPE_NULL Defines a null surface All **Programming Notes** A null surface will be used in instances where an actual surface is not bound. When a write message is generated to a null surface, no actual surface is written to. When a read message (including any sampling engine message) is generated to a null surface, the result is all zeros. Note that a null surface type is allowed to be used with all messages, even if it is not specificially indicated as supported. All of the remaining fields in surface state are ignored for null surfaces, with the following exceptions: • **[DevSNB+]: Width**, **Height**, **Depth**, and **LOD** fields must match the depth buffer's corresponding state for all render target surfaces, including null. • **Surface Format** must be R8G8B8A8_UNORM. The **Surface Type** of a surface used as a render target (accessed via the Data Port's Render Target Write message) must be the same as the **Surface Type** of all other render targets and of the depth buffer (defined in 3DSTATE_DEPTH_BUFFER), unless either the depth buffer or render targets are SURFTYPE_NULL.
	28	**Reserved** Project: All Format: MBZ
	27	**Data Return Format** Project: All Format: U1 enumerated type FormatDesc **For Sampling Engine Surfaces, [DevBW] and [DevCL] only:** This field determines the format of the return data from the sampling engine to the compute engine, but only if the **Data Return Format** field in the message descriptor is set to FLOAT32. This field is ignored for surfaces used by other units. **For Other Surfaces:** This field is ignored. For **[DevCTG+]** Sampling Engine surfaces, the state of this bit is effectively DATA_RETURN_FLOAT32 regardless of its programmed value. **Value** **Name** **Description** **Project** 0h DATA_RETURN_FLOAT32 FLOAT32 data is returned All 1h DATA_RETURN_S1.14 S1.14 fixed point data is returned [DevBW], [DevCL] **Programming Notes** The S1.14 return format is only legal for returning data from normalized (UNORM, or SNORM) map formats where *all* channels have <= 8 bits. *It is not legal to use this format with any floating point or integer map format.* S1.14 return format is only used for SIMD16 and SIMD8 messages from the sampling engine. For SIMD4x2 messages, FLOAT32 format will be used for surfaces specifying S1.14 data return format. Data returned in format S1.14 will be converted to FLOAT32 before reaching the GRF register, thus the state of this bit does not affect the kernel. It is recommended that S1.14 format be used wherever it is legal, as the performance will generally be improved.

SURFACE_STATE		
	26:18	**Surface Format** Project: All Format: U9 FormatDesc Specifies the format of the surface or element within this surface. This field is ignored for all data port messages other than the render target message and streamed vertex buffer write message. Some forms of the media block messages use the surface format. Refer to the table in section 0 for the formats supported and their encodings. **Programming Notes** **Tile Walk** TILEWALK_YMAJOR is UNDEFINED for *render target* formats that have 128 bits-per-element (BPE). YUV (YCRCB) surfaces used as render targets can only be rendered to using 3DPRIM_RECTLIST with even X coordinates on all of its vertices, and the pixel shader cannot kill pixels. If **Number of Multisamples** is set to a value *other than* MULTISAMPLECOUNT_1, this field cannot be set to the following formats: • any format with greater than 64 bits per element • any compressed texture format (BC*) • any YCRCB* format
	17:14	**[DevSNB+]:** Reserved : MBZ (this field has been moved to BLEND_STATE)
	13	**[DevSNB+]:** Reserved : MBZ (this field has been superseded by the **Color Buffer Blend Enable** field in BLEND_STATE)
	12	**Vertical Line Stride** Project: All Format: U1 in lines to skip between logically adjacent lines FormatDesc **For 2D Non-Array Surfaces accessed via the Sampling Engine or Data Port:** Specifies number of lines (0 or 1) to skip between logically adjacent lines – provides support of interleaved (field) surfaces as textures. **For Other Surfaces:** Vertical Line Stride must be zero. **Programming Notes** This bit must not be set if the surface format is a compressed type (BCn*). If this bit is set on a sampling engine surface, texture addess control modes cannot be set to any mode other than TEXCOORDMODE_CLAMP and the mip mode filter must be set to MIPFILTER_NONE.

SURFACE_STATE

	11	**Vertical Line Stride Offset** Project: All Format: U1 in lines of initial offset (when Vertical Line Stride == 1) FormatDesc **For 2D Non-Array Surfaces accessed via the Sampling Engine or Data Port:** Specifies the offset of the initial line from the beginning of the buffer. Ignored when Vertical **Line Stride** is 0. **For Other Surfaces:** Vertical Line Stride Offset must be zero. Errata: project DevSNB Description: **If "Number of Multisamples"** is MULTISAMPLECOUNT 1 and "Vertical **Line Stride"** is 0 Vertical Line Stride Offset must be zero If "**Number of Multisamples"** is any value other than MULTISAMPLECOUNT_1 Vertical Line Stride Offset must be one
	10	**MIP Map Layout Mode** Project: All Format: U1 enumerated type FormatDesc **For 1D and 2D Surfaces and** **For Cube Surfaces** (**[DevILK+]** only): This field specifies which MIP map layout mode is used, whether the map for LOD 1 is stored to the right of the LOD 0 map, or stored below it. See Memory Data Formats for details on the specifics of each layout mode. **For Other Surfaces:** This field is reserved : MBZ **Value** **Name** **Description** **Project** 0h MIPLAYOUT_BELOW All 1h MIPLAYOUT_RIGHT All **Programming Notes** MIPLAYOUT_RIGHT is legal only for 2D non-array surfaces

	9	**Cube Map Corner Mode** Project: All Format: U1 enumerated type FormatDesc **For Cube Surfaces accessed by the Sampling Engine:** When filtering at the corner of cube map one of the four texels does not exist. This field specifies if it gets replaced with the opposite corner texel or the average of all three that exist. **For Other Surfaces:** This field is Reserved : MBZ **Value** **Name** **Description** **Project** 0h CUBE_REPLICATE All 1h CUBE_AVERAGE [DevILK-B+] **Programming Notes** CUBE_AVERAGE may only be selected if all of the **Cube Face Enable** fields are equal to one. **ChromaKey Enable** must not be set in CUBE_AVERAGE mode
	8	**Render Cache Read Write Mode** Project: All Format: U1 enumerated type FormatDesc **For Surfaces accessed via the Data Port to Render Cache:** This field specifies the way Render Cache treats a write request. If unset, Render Cache allocates a write-only cache line for a write miss. If set, Render Cache allocates a read-write cache line for a write miss. **For Surfaces accessed via the Sampling Engine or Data Port to Texture Cache or Data Cache:** This field is reserved : MBZ **Value** **Name** **Description** **Project** 0h Allocating write-only cache for a write miss All 1h Allocating read-write cache for a write miss All **Programming Notes** This field is provided for performance optimization for Render Cache read/write accesses (from DevSNB EU's point of view). **Errata** **Description** **Project** # This field must be set to 0h. [DevBW-A,B]

SURFACE_STATE

	7:6	**Media Boundary Pixel Mode** Project: All Format: U2 enumerated type FormatDesc **For 2D Non-Array Surfaces accessed via the Data Port Media Block Read Message:** This field enables control of which rows are returned on vertical out-of-bounds reads using the Data Port Media Block Read Message. In the description below, frame mode refers to **Vertical Line Stride** = 0, field mode is **Vertical Line Stride** = 1 in which only the even or odd rows are addressable. The frame refers to the entire surface, while the field refers only to the even or odd rows within the surface. Refer to section **Error! Reference source not found.** for more details. **For Other Surfaces:** Reserved : MBZ

Value	Name	Description	Project
0h	NORMAL_MODE	the row returned on an out-of-bound access is the closest row in the frame or field. Rows from the opposite field are never returned.	All
1h	Reserved		All
2h	PROGRESSIVE_FRAME	the row returned on an out-of-bound access is the closest row in the frame, even if in field mode.	[DevCTG+]
3h	INTERLACED_FRAME	in field mode, the row returned on an out-of-bound access is the closest row in the field. In frame mode, even out-of-bound rows return the nearest even row while odd out-of-bound rows return the nearest odd row.	[DevCTG+]

	5:0	**Cube Face Enables** Project: All Format: U6 bit mask of enables FormatDesc **For SURFTYPE_CUBE Surfaces accessed via the Sampling Engine:** Bits 5:0 of this field enable the individual faces of a cube map. Enabling a face indicates that the face is present in the cube map, while disabling it indicates that that face is represented by the texture map's border color. Refer to Memory Data Formats for the correlation between faces and the cube map memory layout. Note that storage for disabled faces must be provided. **For other surfaces:** This field is reserved : MBZ

Value	Name	Description	Project
100000b		-X face	All
010000b		+X face	All

		001000b -Y face All 000100b +Y face All 000010b -Z face All 000001b +Z face All **Programming Notes** When TEXCOORDMODE_CLAMP is used when accessing a cube map, this field must be programmed to 111111b (all faces enabled). This field is ignored unless the **Surface Type** is SURFTYPE_CUBE.
1	31:0	**Surface Base Address** Project: All Format: GraphicsAddress[31:0] FormatDesc Specifies the byte-aligned base address of the surface. **Programming Notes** For SURFTYPE_BUFFER render targets, this field specifies the base address of first element of the surface. The surface is interpreted as a simple array of that single element type. The address must be naturally-aligned to the element size (e.g., a buffer containing R32G32B32A32_FLOAT elements must be 16-byte aligned). For SURFTYPE_BUFFER non-rendertarget surfaces, this field specifies the base address of the first element of the surface, computed in software by adding the surface base address to the byte offset of the element in the buffer. Mipmapped, cube and 3D sampling engine surfaces are stored in a “monolithic” (fixed) format, and only require a single address for the base texture. Linear *render target* surface base addresses must be element-size aligned, for non-YUV surface formats, or a multiple of 2 element-sizes for YUV surface formats. Other linear surfaces have no alignment requirements (byte alignment is sufficient.) **ERRATA [DevSNB-A0]: Linear render target base for element size smaller than a DW, base address must be DW aligned at minimum.** Linear depth buffer surface base addresses must be 64-byte aligned. Note that while render targets (color) can be SURFTYPE_BUFFER, depth buffers cannot. Tiled surface base addresses must be 4KB-aligned. Note that only the offsets from **Surface Base Address** are tiled, **Surface Base Address** itself is not transformed using the tiling algorithm. **[DevCTG+]:** For tiled surfaces, the actual start of the surface can be offset from the **Surface Base Address** by the **X Offset** and **Y Offset** fields. Certain message types used to access surfaces have more stringent alignment requirements. Please refer to the specific message documentation for additional restrictions.

SURFACE_STATE		
2	31:19	**Height** Project: All Format: U13 FormatDesc Range SURFTYPE_1D: must be zero SURFTYPE_2D: height of surface – 1 (y/v dimension) [0,8191] SURFTYPE_3D: height of surface – 1 (y/v dimension) [0,2047] SURFTYPE_CUBE: height of surface – 1 (y/v dimension) [0,8191] SURFTYPE_BUFFER: contains bits [19:7] of the number of entries in the buffer – 1 [0,8191] This field specifies the height of the surface. If the surface is MIP-mapped, this field contains the height of the base MIP level. For buffers, this field specifies a portion of the buffer size. **Programming Notes** For buffer surfaces, the number of entries in the buffer ranges from 1 to 2^{27}. After subtracting one from the number of entries, software must place the fields of the resulting 27-bit value into the **Height**, **Width**, and **Depth** fields as indicated, right-justified in each field. Unused upper bits must be set to zero. If **Vertical Line Stride** is 1, this field indicates the height of the field, not the height of the frame The **Height** of a render target must be the same as the **Height** of the other render targets and the depth buffer (defined in 3DSTATE_DEPTH_BUFFER), unless **Surface Type** is SURFTYPE_1D or SURFTYPE_2D with **Depth** = 0 (non-array) and **LOD** = 0 (non-mip mapped).

SURFACE_STATE		
	18:6	**Width** Project: All Format: U13 FormatDesc Range SURFTYPE_1D: width of surface – 1 (x/u dimension) [0,8191] SURFTYPE_2D: width of surface – 1 (x/u dimension) [0,8191] SURFTYPE_3D: width of surface – 1 (x/u dimension) [0,2047] SURFTYPE_CUBE: width of surface – 1 (x/u dimension) [0,8191] SURFTYPE_BUFFER: contains bits [6:0] of the number of entries in the buffer – 1 [0,127] This field specifies the width of the surface. If the surface is MIP-mapped, this field specifies the width of the base MIP level. The width is specified in units of pixels or texels. For buffers, this field specifies a portion of the buffer size. For surfaces accessed with the Media Block Read/Write message, this field is in units of DWords. **Programming Notes** For surface types other than SURFTYPE_BUFFER, the Width specified by this field must be less than or equal to the surface pitch (specified in bytes via the **Surface Pitch** field). For cube maps, Width must be set equal to the Height. For MONO8 textures, Width must be a multiple of 32 texels. The **Width** of a render target must be the same as the **Width** of the other render target(s) and the depth buffer (defined in 3DSTATE_DEPTH_BUFFER), unless **Surface Type** is SURFTYPE_1D or SURFTYPE_2D with **Depth** = 0 (non-array) and **LOD** = 0 (non-mip mapped). The **Width** of a render target with YUV surface format must be a multiple of 2.

	5:2	**MIP Count / LOD** Project: All Format: Sampling Engine Surfaces: U4 in (LOD units – 1) FormatDesc Render Target Surfaces: U4 in LOD units Range Sampling Engine Surfaces: [0,13] representing [1,14] MIP levels Render Target Surfaces: [0,13] representing LOD Other Surfaces: [0] **For Sampling Engine Surfaces:** This field indicates the number of MIP levels allowed to be accessed starting at **Surface Min LOD**, which must be less than or equal to the number of MIP levels actually stored in memory for this surface. Force the mip map access to be between the mipmap specified by the integer bits of the Min LOD and the ceiling of the value specified here. **For Render Target Surfaces:** This field defines the MIP level that is currently being rendered into. This is the absolute MIP level on the surface and is not relative to the **Surface Min LOD** field, which is ignored for render target surfaces. **For Other Surfaces:** This field is reserved : MBZ **Programming Notes** The **LOD** of a render target must be the same as the **LOD** of the other render target(s) and of the depth buffer (defined in 3DSTATE_DEPTH_BUFFER). For render targets with YUV surface formats, the **LOD** must be zero.

Value	Name	Description	Project
0h	Disable	Desc	All
1h	Enable	Desc	All

SURFACE_STATE

	1:0	**Render Target Rotation** Project: All Format: U2 enumerated type FormatDesc **For Render Target Surfaces:** This field specifies the rotation of this render target surface when being written to memory. **For Other Surfaces:** This field is ignored. **Programming Notes** Rotation is not supported for render targets of any type other than simple, non-mip-mapped, non-array 2D surfaces. The surface must be using tiled with X major. **Width** and **Height** fields apply to the dimensions of the surface before rotation. For 90 and 270 degree rotated surfaces, the **Height** (rather than the **Width**) must be less than or equal to the **Surface Pitch** (specified in bytes). For 90 and 270 degree rotated surfaces, the actual **Height** and **Width** of the surface in pixels (not the field value which is decremented) must both be even. Rotation is supported only for surfaces with the following surface formats: B5G6R5_UNORM, B5G6R5_UNORM_SRGB, R8G8B8[A|X]8_UNORM, R8G8B8[A|X]8_UNORM_SRGB, B8G8R8[A|X]8_UNORM, B8G8R8[A|X]8_UNORM_SRGB, B10G10R10[A|X]2_UNORM, B10G10R10A2_UNORM_SRGB, R10G10B10A2_UNORM, R10G10B10A2_UNORM_SRGB, R16G16B16A16_FLOAT, R16G16B16X16_FLOAT

Value	Name	Description	Project
0h	RTROTATE_0DEG	No rotation (0 degrees)	All
1h	RTROTATE_90DEG	Rotate by 90 degrees	All
2h	Reserved		All
3h	RTROTATE_270DEG	Rotate by 270 degrees	All

SURFACE_STATE

3	31:21	**Depth** Project: All Format: U11 FormatDesc Range SURFTYPE_1D: number of array elements – 1 [0,511] SURFTYPE_2D: number of array elements – 1 [0,511] SURFTYPE_3D: depth of surface – 1 (z/r dimension) [0,2047] SURFTYPE_CUBE: number of array elements – 1 [see programming notes for range] SURFTYPE_BUFFER: contains bits [26:20] of the number of entries in the buffer – 1 [0,127] This field specifies the total number of levels for a volume texture or the number of array elements allowed to be accessed starting at the **Minimum Array Element** for arrayed surfaces. If the volume texture is MIP-mapped, this field specifies the depth of the base MIP level. For buffers, this field specifies a portion of the buffer size. **Programming Notes** The **Depth** of a render target must be the same as the **Depth** of the other render target(s) and of the depth buffer (defined in 3DSTATE_DEPTH_BUFFER). For SURFTYPE_CUBE: **[DevSNB+]:** for **Sampling Engine Surfaces**, the range of this field is [0,84], indicating the number of cube array elements (equal to the number of underlying 2D array elements *divided by 6*). For other surfaces, this field must be zero.
	20	**Reserved** Project: All Format: MBZ
	19:3	**Surface Pitch** Project: All Format: U17 pitch in (#Bytes – 1) FormatDesc Range For surfaces of type SURFTYPE_BUFFER: [0,2047] -> [1B, 2048B] For surfaces of type SURFTYPE_STRBUF: [0,2047] -> [1B, 2048B] For other linear surfaces: [0, 524287] -> [1B, 512KB] For X-tiled surface: [511, 524287] –> [512B, 512KB] = [1 tile, 1024 tiles] For Y-tiled surfaces: [127, 524287]->[128B, 512KB] = [1 tile, 4096 tiles] This field specifies the surface pitch in (#Bytes - 1). For surfaces of type SURFTYPE_BUFFER, this field indicates the size of the structure. **Programming Notes** For linear *render target* surfaces, the pitch must be a multiple of the element size for non-YUV surface formats. Pitch must be a multiple of 2 * element size for YUV surface formats. For other linear surfaces, the pitch can be any multiple of bytes. For tiled surfaces, the pitch must be a multiple of the tile width.

SURFACE_STATE

	2	**Reserved** Project: All Format: MBZ
	1	**Tiled Surface** Project: All Format: U1 enumerated type FormatDesc This field specifies whether the surface is tiled. **Value** **Name** **Description** **Project** 0h FALSE Linear surface All 1h TRUE Tiled surface All **Programming Notes** Linear surfaces can be mapped to Main Memory (uncached) or System Memory (cacheable, snooped). Tiled surfaces can only be mapped to Main Memory. The corresponding cache(s) must be invalidated before a previously accessed surface is accessed again with an altered state of this bit. If **Surface Type** is SURFTYPE_BUFFER, this field must be FALSE (buffers are supported only in linear memory) If the target cache via the Data Port is the Data Cache, this field must be disabled (zero). The data cache only supports access to linear memory. If **Surface Type** is SURFTYPE_NULL, this field must be TRUE [DevSNB+]: For multi-sample render targets, this field must be 1. MSRTs can only be tiled.

	0	**Tile Walk** Project: All Format: U1 enumerated type FormatDesc This field specifies the type of memory tiling (XMajor or YMajor) employed to tile this surface. See *Memory Interface Functions* for details on memory tiling and restrictions. **Value** **Name** **Description** **Project** 0h TILEWALK_XMAJOR X major tiling All 1h TILEWALK_YMAJOR Y major tiling All **Programming Notes** Refer to *Memory Data Formats* for restrictions on *TileWalk* direction for the various buffer types. (Of particular interest is the fact that YMAJOR tiling is **not** supported for display/overlay buffers). The corresponding cache(s) must be invalidated before a previously accessed surface is accessed again with an altered state of this bit. Use of TILEWALK_YMAJOR is UNDEFINED for render target formats that have 128 bits-per-element (BPE). This field is ignored when the surface is linear. **Errata:** [DevSNB] Set Tile Walk to TILEWALK_XMAJOR if Tiled Surface set to False
4	31:28	**Surface Min LOD** Project: All Format: U4 in LOD units FormatDesc Range [0,13] **For Sampling Engine Surfaces:** This field indicates the most detailed LOD that can be accessed as part of this surface. This field is added to the delivered LOD (sample_l, ld, or resinfo message types) before it is used to address the surface. **For Other Surfaces:** This field is ignored. **Programming Notes** This field must be zero if the **Surface Format** is MONO8 **[DevBW-A,B]:** this field must be zero

SURFACE_STATE		
	27:17	**Minimum Array Element** Project: All Format: U11 FormatDesc Range 1D/2D/cube surfaces: [0,511] 3D surfaces: [0,2047] **For Sampling Engine and Render Target 1D and 2D Surfaces:** This field indicates the minimum array element that can be accessed as part of this surface. This field is added to the delivered array index before it is used to address the surface. **For Render Target 3D Surfaces:** This field indicates the minimum 'R' coordinate on the LOD currently being rendered to. This field is added to the delivered array index before it is used to address the surface. **For Sampling Engine Cube Surfaces on [DevSNB+] only:** This field indicates the minimum array element in the *underlying 2D surface array* that can be accessed as part of this surface (the cube array index is multipled by 6 to compute this value, although this field is *not* restricted to only multiples of 6). This field is added to the delivered array index before it is used to address the surface. **For Other Surfaces:** This field must be set to zero.
	16:8	**Render Target View Extent** Project: All Format: U9 FormatDesc Range [0,511] to indicate extent of [1,512] **For Render Target 3D Surfaces:** This field indicates the extent of the accessible 'R' coordinates minus 1 on the LOD currently being rendered to. **For Render Target 1D and 2D Surfaces:** This field must be set to the same value as the **Depth** field. **For Other Surfaces:** This field is ignored.
	7	**Reserved** Project: All Format: MBZ

SURFACE_STATE

	6:4	**Number of Multisamples** Project: [DevSNB+] Format: U3 enumerated type FormatDesc This field indicates the number of multisamples on the surface. **[Pre-DevSNB]:** Reserved : MBZ **Value** **Name** **Description** **Project** 0h MULTISAMPLECOUNT_1 All 1h Reserved All 2h MULTISAMPLECOUNT_4 All 3h-7h Reserved All **Programming Notes** If this field is any value other than MULTISAMPLECOUNT_1 the following restrictions apply: • the **Surface Type** must be SURFTYPE_2D • For sampling engine messages other than "ld", the U and V addresses for all pixels must be within the following range: o U * width – 0.5 ≥ 0 and ≤ (width -2) o V * height – 0.5 ≥ 0 and ≤ (height -2)
	3	**Reserved** Project: All Format: MBZ
	2:0	**Multisample Position Palette Index** Project: [DevSNB+] Format: U3 FormatDesc Range [0,7] This field indicates the index into the sample position palette that the multisampled surface is using. This field is only used as a return value for the sampleinfo message, and is otherwise not used by hardware.

SURFACE_STATE

5	31:25	**X Offset** Project: All Format: PixelOffset[8:2] FormatDesc Range TileX surfaces: [0,ceil(512/BytesPerElement)4] in multiples of 4 (low 2 bits missing) TileY surfaces: [0,ceil(128/BytesPerElement)-4] in multiples of 4 (low 2 bits missing) This field specifies the horizontal offset in pixels from the **Surface Base Address** to the start (origin) of the surface. This field effectively loosens the alignment restrictions on the origin of tiled surfaces. Previously, tiled surface origin was (by definition) located at the base address, and thus needed to satisfy the 4KB base address alignment restriction. Now the origin can be specified at a finer (4-wide x 2-high pixel) resolution. **Programming Notes** For linear surfaces, this field must be zero For surfaces accessed with the Data Port Media Block Read/Write message, the pixel size is assumed to be 32 bits in width For **Surface Format** with other than 8, 16, 32, 64, or 128 bits per pixel, this field must be zero. If **Render Target Rotation** is set to other than RTROTATE_0DEG, this field must be zero.
	24	**Surface Vertical Alignment** Project: [DevSNB+] Format: U1 enumerated type FormatDesc **For Sampling Engine Uncompressed and Render Target Surfaces:** This field specifies the vertical alignment requirement for the surface. Refer to the "Memory Data Formats" chapter for details on how this field changes the layout of the surface in memory. This field applies to surface formats other than compressed formats. **For Other Surfaces:** This field is ignored. **Value** / **Name** / **Description** / **Project** 0h / VALIGN_2 / Vertical alignment factor j = 2 / All 1h / VALIGN_4 / Vertical alignment factor j = 4 / All **Programming Notes** This field must be set to VALIGN_2 if the **Surface Format** is 96 bits per element (BPE). [DevSNB]: Value of 1 is not supported for format YCRCB_NORMAL (0x182), YCRCB_SWAPUVY (0x183), YCRCB_SWAPUV (0x18f), YCRCB_SWAPY (0x190)

SURFACE_STATE		
	23:20	**Y Offset** Project: All Format: RowOffset[4:1] FormatDesc Range TileX surfaces: [0,6] in multiples of 2 (low bit missing) TileY surfaces: [0,30] in multiples of 2 (low bit missing) This field specifies the vertical offset in rows from the **Surface Base Address** to the start of the surface. (See additional description in the **X Offset** field) **Programming Notes** For linear surfaces, this field must be zero. For render targets in which the Render Target Array Index is not zero, this field must be zero. For **Surface Format** with other than 8, 16, 32, 64, or 128 bits per pixel, this field must be zero. If **Render Target Rotation** is set to other than RTROTATE_0DEG, this field must be zero. **[DevILK+]:** For surfaces accessed in field mode (**Vertical Line Stride** = 1 or equivalent Media Block Read/Write message override), this field must be set to a multiple of 4.
	19:16	**Surface Object Control State (MEMORY_OBJECT_CONTROL_STATE)** Project: [DevSNB+] Format: MEMORY_OBJECT_CONTROL_STATE FormatDesc Specifies the memory object control state for this surface. **DevSNB A Step Erratum**: When a surface is mapped through MT (Sampler Cache) either from Sampler or from Read Data Port, the Cacheability Control bits [1:0] are forced to zero by hardware. Thus it is solely relies on the control from GTT entries
	15:0	**Reserved** Project: All Format: MBZ

2.11.2.1.2 Surface Formats

The following table indicates the supported surface formats and the 9-bit encoding for each. Note that some of these formats are used not only by the Sampling Engine, but also by the Data Port and the Vertex Fetch unit.

Support of each format and capability is as follows:

Y	supported on all products
Y*	supported only on **[DevCTG+]**
Y+	supported only on **[DevCTG-B+]**
Y~	supported only on **[DevILK+]**
Y^	supported only on **[DevSNB+]**

Sampling Engine	Sampling Engine Filtering	Sampling Engine Shadow Map	Sampling Engine Chroma Key	Render Target	Alpha Blend Render Target	Input Vertex Buffer	Streamed Output Vertex Buffers	Color Processing	Surface Format Encoding (Hex)	Format Name	Bits Per Element (BPE)
Y	Y~			Y	Y	Y	Y		000	R32G32B32A32_FLOAT	128**
Y				Y		Y	Y		001	R32G32B32A32_SINT	128**
Y				Y		Y	Y		002	R32G32B32A32_UINT	128**
						Y			003	R32G32B32A32_UNORM	128
						Y			004	R32G32B32A32_SNORM	128
						Y			005	R64G64_FLOAT	128
Y	Y~								006	R32G32B32X32_FLOAT	128
						Y			007	R32G32B32A32_SSCALED	128
						Y			008	R32G32B32A32_USCALED	128
Y	Y~					Y	Y		040	R32G32B32_FLOAT	96

Sampling Engine	Sampling Engine Filtering	Sampling Engine Shadow Map	Sampling Engine Chroma Key	Render Target	Alpha Blend Render Target	Input Vertex Buffer	Streamed Output Vertex Buffers	Color Processing	Surface Format Encoding (Hex)	Format Name	Bits Per Element (BPE)
Y						Y	Y		041	R32G32B32_SINT	96
Y						Y	Y		042	R32G32B32_UINT	96
						Y			043	R32G32B32_UNORM	96
						Y			044	R32G32B32_SNORM	96
						Y			045	R32G32B32_SSCALED	96
						Y			046	R32G32B32_USCALED	96
Y	Y			Y	Y+	Y		Y^	080	R16G16B16A16_UNORM	64
Y	Y			Y	Y^	Y			081	R16G16B16A16_SNORM	64
Y				Y		Y			082	R16G16B16A16_SINT	64
Y				Y		Y			083	R16G16B16A16_UINT	64
Y	Y			Y	Y	Y			084	R16G16B16A16_FLOAT	64
Y	Y~			Y	Y	Y	Y		085	R32G32_FLOAT	64
Y				Y		Y	Y		086	R32G32_SINT	64
Y				Y		Y	Y		087	R32G32_UINT	64
Y	Y~	Y							088	R32_FLOAT_X8X24_TYPELESS	64
Y									089	X32_TYPELESS_G8X24_UINT	64
Y	Y~								08A	L32A32_FLOAT	64
						Y			08B	R32G32_UNORM	64
						Y			08C	R32G32_SNORM	64
						Y			08D	R64_FLOAT	64

Sampling Engine	Sampling Engine Filtering	Sampling Engine Shadow Map	Sampling Engine Chroma Key	Render Target	Alpha Blend Render Target	Input Vertex Buffer	Streamed Output Vertex Buffers	Color Processing	Surface Format Encoding (Hex)	Format Name	Bits Per Element (BPE)
Y	Y								08E	R16G16B16X16_UNORM	64
Y	Y								08F	R16G16B16X16_FLOAT	64
Y	Y~								090	A32X32_FLOAT	64
Y	Y~								091	L32X32_FLOAT	64
Y	Y~								092	I32X32_FLOAT	64
						Y			093	R16G16B16A16_SSCALED	64
						Y			094	R16G16B16A16_USCALED	64
						Y			095	R32G32_SSCALED	64
						Y			096	R32G32_USCALED	64
Y	Y		Y	Y	Y	Y		Y^	0C0	B8G8R8A8_UNORM	32
Y	Y			Y	Y				0C1	B8G8R8A8_UNORM_SRGB	32
Y	Y			Y	Y	Y		Y^	0C2	R10G10B10A2_UNORM	32
Y	Y							Y^	0C3	R10G10B10A2_UNORM_SRGB	32
Y				Y		Y			0C4	R10G10B10A2_UINT	32
Y	Y					Y			0C5	R10G10B10_SNORM_A2_UNORM	32
Y	Y			Y	Y	Y		Y^	0C7	R8G8B8A8_UNORM	32
Y	Y			Y	Y			Y^	0C8	R8G8B8A8_UNORM_SRGB	32
Y	Y			Y	Y^	Y			0C9	R8G8B8A8_SNORM	32
Y				Y		Y			0CA	R8G8B8A8_SINT	32
Y				Y		Y			0CB	R8G8B8A8_UINT	32

Sampling Engine	Sampling Engine Filtering	Sampling Engine Shadow Map	Sampling Engine Chroma Key	Render Target	Alpha Blend Render Target	Input Vertex Buffer	Streamed Output Vertex Buffers	Color Processing	Surface Format Encoding (Hex)	Format Name	Bits Per Element (BPE)
Y	Y			Y	Y+	Y			0CC	R16G16_UNORM	32
Y	Y			Y	Y^	Y			0CD	R16G16_SNORM	32
Y				Y		Y			0CE	R16G16_SINT	32
Y				Y		Y			0CF	R16G16_UINT	32
Y	Y			Y	Y	Y			0D0	R16G16_FLOAT	32
Y	Y			Y	Y			Y^	0D1	B10G10R10A2_UNORM	32
Y	Y			Y	Y			Y^	0D2	B10G10R10A2_UNORM_SRGB	32
Y	Y			Y	Y	Y			0D3	R11G11B10_FLOAT	32
Y				Y		Y	Y		0D6	R32_SINT	32
Y				Y		Y	Y		0D7	R32_UINT	32
Y	Y~	Y		Y	Y	Y	Y		0D8	R32_FLOAT	32
Y	Y~	Y							0D9	R24_UNORM_X8_TYPELESS	32
Y									0DA	X24_TYPELESS_G8_UINT	32
Y	Y								0DF	L16A16_UNORM	32
Y	Y~	Y							0E0	I24X8_UNORM	32
Y	Y~	Y							0E1	L24X8_UNORM	32
Y	Y~	Y							0E2	A24X8_UNORM	32
Y	Y~	Y							0E3	I32_FLOAT	32
Y	Y~	Y							0E4	L32_FLOAT	32
Y	Y~	Y							0E5	A32_FLOAT	32

Sampling Engine	Sampling Engine Filtering	Sampling Engine Shadow Map	Sampling Engine Chroma Key	Render Target	Alpha Blend Render Target	Input Vertex Buffer	Streamed Output Vertex Buffers	Color Processing	Surface Format Encoding (Hex)	Format Name	Bits Per Element (BPE)
Y	Y		Y					Y^	0E9	B8G8R8X8_UNORM	32
Y	Y								0EA	B8G8R8X8_UNORM_SRGB	32
Y	Y								0EB	R8G8B8X8_UNORM	32
Y	Y								0EC	R8G8B8X8_UNORM_SRGB	32
Y	Y								0ED	R9G9B9E5_SHAREDEXP	32
Y	Y								0EE	B10G10R10X2_UNORM	32
Y	Y								0F0	L16A16_FLOAT	32
						Y			0F1	R32_UNORM	32
						Y			0F2	R32_SNORM	32
						Y			0F3	R10G10B10X2_USCALED	32
						Y			0F4	R8G8B8A8_SSCALED	32
						Y			0F5	R8G8B8A8_USCALED	32
						Y			0F6	R16G16_SSCALED	32
						Y			0F7	R16G16_USCALED	32
						Y			0F8	R32_SSCALED	32
						Y			0F9	R32_USCALED	32
									0FA	R8B8G8A8_UNORM	32
Y	Y		Y	Y	Y				100	B5G6R5_UNORM	16
Y	Y			Y	Y				101	B5G6R5_UNORM_SRGB	16
Y	Y		Y	Y	Y				102	B5G5R5A1_UNORM	16

Sampling Engine	Sampling Engine Filtering	Sampling Engine Shadow Map	Sampling Engine Chroma Key	Render Target	Alpha Blend Render Target	Input Vertex Buffer	Streamed Output Vertex Buffers	Color Processing	Surface Format Encoding (Hex)	Format Name	Bits Per Element (BPE)
Y	Y			Y	Y				103	B5G5R5A1_UNORM_SRGB	16
Y	Y		Y	Y	Y				104	B4G4R4A4_UNORM	16
Y	Y			Y	Y				105	B4G4R4A4_UNORM_SRGB	16
Y	Y			Y	Y	Y			106	R8G8_UNORM	16
Y	Y		Y	Y	Y^	Y			107	R8G8_SNORM	16
Y				Y		Y			108	R8G8_SINT	16
Y				Y		Y			109	R8G8_UINT	16
Y	Y	Y		Y	Y+	Y			10A	R16_UNORM	16
Y	Y			Y	Y^	Y			10B	R16_SNORM	16
Y				Y		Y			10C	R16_SINT	16
Y				Y		Y			10D	R16_UINT	16
Y	Y			Y	Y	Y			10E	R16_FLOAT	16
Y~	Y~								10F	A8P8_UNORM [palette0]	16
Y~	Y~								110	A8P8_UNORM [palette1]	16
Y	Y	Y							111	I16_UNORM	16
Y	Y	Y							112	L16_UNORM	16
Y	Y	Y							113	A16_UNORM	16
Y	Y		Y						114	L8A8_UNORM	16
Y	Y	Y							115	I16_FLOAT	16
Y	Y	Y							116	L16_FLOAT	16

Sampling Engine	Sampling Engine Filtering	Sampling Engine Shadow Map	Sampling Engine Chroma Key	Render Target	Alpha Blend Render Target	Input Vertex Buffer	Streamed Output Vertex Buffers	Color Processing	Surface Format Encoding (Hex)	Format Name	Bits Per Element (BPE)
Y	Y	Y							117	A16_FLOAT	16
Y*	Y*								118	L8A8_UNORM_SRGB	16
Y	Y		Y						119	R5G5_SNORM_B6_UNORM	16
				Y	Y				11A	B5G5R5X1_UNORM	16
				Y	Y				11B	B5G5R5X1_UNORM_SRGB	16
						Y			11C	R8G8_SSCALED	16
						Y			11D	R8G8_USCALED	16
						Y			11E	R16_SSCALED	16
						Y			11F	R16_USCALED	16
Y~	Y~								122	P8A8_UNORM [palette0]	16
Y~	Y~								123	P8A8_UNORM [palette1]	16
Y	Y		Y*	Y	Y	Y			140	R8_UNORM	8
Y	Y			Y	Y^	Y			141	R8_SNORM	8
Y				Y		Y			142	R8_SINT	8
Y				Y		Y			143	R8_UINT	8
Y	Y		Y	Y	Y				144	A8_UNORM	8
Y	Y								145	I8_UNORM	8
Y	Y		Y						146	L8_UNORM	8
Y	Y								147	P4A4_UNORM [palette0]	8
Y	Y								148	A4P4_UNORM [palette0]	8

Sampling Engine	Sampling Engine Filtering	Sampling Engine Shadow Map	Sampling Engine Chroma Key	Render Target	Alpha Blend Render Target	Input Vertex Buffer	Streamed Output Vertex Buffers	Color Processing	Surface Format Encoding (Hex)	Format Name	Bits Per Element (BPE)
						Y			149	R8_SSCALED	8
						Y			14A	R8_USCALED	8
Y*	Y*								14B	P8_UNORM [palette0]	8
Y*	Y*								14C	L8_UNORM_SRGB	8
Y+	Y+								14D	P8_UNORM [palette1]	8
Y+	Y+								14E	P4A4_UNORM [palette1]	8
Y+	Y+								14F	A4P4_UNORM [palette1]	8
Y*	Y*								180	DXT1_RGB_SRGB	0
Y	Y								181	R1_UNORM/R1_UINT	1
Y	Y		Y	Y				Y^	182	YCRCB_NORMAL	0
Y	Y		Y	Y				Y^	183	YCRCB_SWAPUVY	0
Y*	Y*								184	P2_UNORM [palette0]	2
Y+	Y+								185	P2_UNORM [palette1]	2
Y	Y		Y						186	BC1_UNORM (DXT1)	0
Y	Y		Y						187	BC2_UNORM (DXT2/3)	0
Y	Y		Y						188	BC3_UNORM (DXT4/5)	0
Y	Y								189	BC4_UNORM	0
Y	Y								18A	BC5_UNORM	0
Y	Y								18B	BC1_UNORM_SRGB (DXT1_SRGB)	0
Y	Y								18C	BC2_UNORM_SRGB (DXT2/3_SRGB)	0

Sampling Engine	Sampling Engine Filtering	Sampling Engine Shadow Map	Sampling Engine Chroma Key	Render Target	Alpha Blend Render Target	Input Vertex Buffer	Streamed Output Vertex Buffers	Color Processing	Surface Format Encoding (Hex)	Format Name	Bits Per Element (BPE)
Y	Y								18D	BC3_UNORM_SRGB (DXT4/5_SRGB)	0
Y									18E	MONO8	1
Y	Y			Y				Y^	18F	YCRCB_SWAPUV	0
Y	Y			Y				Y^	190	YCRCB_SWAPY	0
Y	Y								191	DXT1_RGB	0
Y	Y								192	FXT1	0
						Y			193	R8G8B8_UNORM	24
						Y			194	R8G8B8_SNORM	24
						Y			195	R8G8B8_SSCALED	24
						Y			196	R8G8B8_USCALED	24
						Y			197	R64G64B64A64_FLOAT	256
						Y			198	R64G64B64_FLOAT	192
Y	Y								199	BC4_SNORM	0
Y	Y								19A	BC5_SNORM	0
Y~	Y~					Y^			19B	R16G16B16_FLOAT	48
						Y			19C	R16G16B16_UNORM	48
						Y			19D	R16G16B16_SNORM	48
						Y			19E	R16G16B16_SSCALED	48
						Y			19F	R16G16B16_USCALED	48
									1A8	R8G8B8_UNORM_SRGB	24

Sampling Engine	Sampling Engine Filtering	Sampling Engine Shadow Map	Sampling Engine Chroma Key	Render Target	Alpha Blend Render Target	Input Vertex Buffer	Streamed Output Vertex Buffers	Color Processing	Surface Format Encoding (Hex)	Format Name	Bits Per Element (BPE)
									1A1	BC6H_SF16	0
									1A2	BC7_UNORM	0
									1A3	BC7_UNORM_SRGB	0
									1A4	BC6H_UF16	0
									1A5	PLANAR_420_8†	0
									1FF	RAW	0

** **Note**: 128 BPE Formats cannot be Tiled Y when used as render targets

† For the PLANAR_420_8 format, the **TCX** and **TCY Address Control Modes** in SAMPLER_STATE must be set to TEXCOORDMODE_CLAMP and the **Height** and **Width** fields in SURFACE_STATE must indicate dimensions that are a multiple of 4 pixels.

NOTE: "RAW" is supported only with buffers and structured buffers accessed via the untyped surface read/write and untyped atomic operation messages, which do not have a column in the table.

2.11.2.1.3 Sampler Output Channel Mapping

The following table indicates the mapping of the channels from the surface to the channels output from the sampling engine. Formats with all four channels (R/G/B/A) in their name map each surface channel to the corresponding output, thus those formats are not shown in this table.

Surface Format Name	R	G	B	A
R32G32B32X32_FLOAT	R	G	B	1.0
R32G32B32_FLOAT	R	G	B	1.0
R32G32B32_SINT	R	G	B	1.0
R32G32B32_UINT	R	G	B	1.0
R32G32_FLOAT	R	G	1.0	1.0

Surface Format Name	R	G	B	A
	R	G	0.0	1.0
R32G32_SINT	R	G	0.0	1.0
R32G32_UINT	R	G	0.0	1.0
R32_FLOAT_X8X24_TYPELESS	R	0.0	0.0	1.0
X32_TYPELESS_G8X24_UINT	0.0	G	0.0	1.0
L32A32_FLOAT	L	L	L	A
R16G16B16X16_UNORM	R	G	B	1.0
R16G16B16X16_FLOAT	R	G	B	1.0
A32X32_FLOAT	0.0	0.0	0.0	A
L32X32_FLOAT	L	L	L	1.0
I32X32_FLOAT	I	I	I	I
R16G16_UNORM	R	G	1.0	1.0
	R	G	0.0	1.0
R16G16_SNORM	R	G	1.0	1.0
	R	G	0.0	1.0
R16G16_SINT	R	G	0.0	1.0
R16G16_UINT	R	G	0.0	1.0
R16G16_FLOAT	R	G	1.0	1.0
	R	G	0.0	1.0
R11G11B10_FLOAT	R	G	B	1.0
R32_SINT	R	0.0	0.0	1.0
R32_UINT	R	0.0	0.0	1.0
R32_FLOAT	R	1.0	1.0	1.0
	R	0.0	0.0	1.0
R24_UNORM_X8_TYPELESS	R	0.0	0.0	1.0
X24_TYPELESS_G8_UINT	0.0	G	0.0	1.0

Surface Format Name	R	G	B	A
L16A16_UNORM	L	L	L	A
I24X8_UNORM	I	I	I	I
L24X8_UNORM	L	L	L	1.0
A24X8_UNORM	0.0	0.0	0.0	A
I32_FLOAT	I	I	I	I
L32_FLOAT	L	L	L	1.0
A32_FLOAT	0.0	0.0	0.0	A
B8G8R8X8_UNORM	R	G	B	1.0
B8G8R8X8_UNORM_SRGB	R	G	B	1.0
R8G8B8X8_UNORM	R	G	B	1.0
R8G8B8X8_UNORM_SRGB	R	G	B	1.0
R9G9B9E5_SHAREDEXP	R	G	B	1.0
B10G10R10X2_UNORM	R	G	B	1.0
L16A16_FLOAT	L	L	L	A
B5G6R5_UNORM	R	G	B	1.0
B5G6R5_UNORM_SRGB	R	G	B	1.0
R8G8_UNORM	R	G	1.0	1.0
	R	G	0.0	1.0
R8G8_SNORM	R	G	1.0	1.0
	R	G	0.0	1.0
R8G8_SINT	R	G	0.0	1.0
R8G8_UINT	R	G	0.0	1.0
R16_UNORM	R	0.0	0.0	1.0
R16_SNORM	R	0.0	0.0	1.0
R16_SINT	R	0.0	0.0	1.0
R16_UINT	R	0.0	0.0	1.0

Surface Format Name	R	G	B	A
R16_FLOAT	R	1.0	1.0	1.0
	R	0.0	0.0	1.0
I16_UNORM	I	I	I	I
L16_UNORM	L	L	L	1.0
A16_UNORM	0.0	0.0	0.0	A
L8A8_UNORM	L	L	L	A
I16_FLOAT	I	I	I	I
L16_FLOAT	L	L	L	1.0
A16_FLOAT	0.0	0.0	0.0	A
R5G5_SNORM_B6_UNORM	R	G	B	1.0
R8_UNORM	R	0.0	0.0	1.0
R8_SNORM	R	0.0	0.0	1.0
R8_SINT	R	0.0	0.0	1.0
R8_UINT	R	0.0	0.0	1.0
A8_UNORM	0.0	0.0	0.0	A
I8_UNORM	I	I	I	I
L8_UNORM	L	L	L	1.0
L8_UNORM_SRGB	L	L	L	1.0
R1_UNORM/R1_UINT	R	0.0	0.0	1.0
YCRCB_NORMAL	Cr	Y	Cb	1.0
YCRCB_SWAPUVY	Cr	Y	Cb	1.0
BC4_UNORM	R	0.0	0.0	1.0
BC5_UNORM	R	G	0.0	1.0
YCRCB_SWAPUV	Cr	Y	Cb	1.0
YCRCB_SWAPY	Cr	Y	Cb	1.0
DXT1_RGB	R	G	B	1.0

Surface Format Name	R	G	B	A
DXT1_RGB_SRGB	R	G	B	1.0
BC4_SNORM	R	0.0	0.0	1.0
BC5_SNORM	R	G	0.0	1.0

2.11.3 SAMPLER_STATE

SAMPLER_STATE has three different formats, depending on the message type used. The sample_8x8 and deinterlace messages use a different format of SAMPLER_STATE as detailed in the corresponding sections.

2.11.3.1 Sampler State for most messages

2.11.3.1.1 SAMPLER_STATE [DevSNB]

SAMPLER_STATE

Project: [DevSNB]

This is the normal sampler state used by all messages that use SAMPLER_STATE except *sample_8x8* and *deinterlace*. The sampler state is stored as an array of up to 16 elements, each of which contains the dwords described here. The start of each element is spaced 4 dwords apart. The first element of the sampler state array is aligned to a 32-byte boundary.

DWord	Bit	Description
0	31	**Sampler Disable** Project: All Format: Disable FormatDesc This field allows the sampler to be disabled. If disabled, all output channels will return 0.
	30	**Reserved** Project: All Format: MBZ
	29	**Reserved**
	28	**LOD PreClamp Enable** Project: All Format: U1 enumerated type FormatDesc When enabled, the computed LOD is clamped to [max,min] mip level *before* the mag-vs-min determination is performed. This is how the OpenGL API currently performs min/mag determination, and therefore it is expected that an OpenGL driver would need to set this bit. **Value** **Name** **Description** **Project** 0h Reserved All 1h OGL OGL Mode (LOD PreClamp enabled) All

SAMPLER_STATE

	27	**Min and Mag State Not Equal** Project: [DevSNB] Format: U1 enumerated type FormatDesc Indicates if state is not the same for min and mag modes. Must be set to 1 if any of the following are true: • **Mag Mode Filter** and **Min Mode Filter** are not the same • **Address Rounding Enable:** U address mag filter and U address min filter are not the same • **Address Rounding Enable:** V address mag filter and V address min filter are not the same **Address Rounding Enable:** R address mag filter and R address min filter are not the same

Value	Name	Description	Project
0h	MIN_MAG_EQ	Min and Mag state are equal	**[DevSNB]**
1h	MIN_MAG_NEQ	Min and Mag state are not equal	**[DevSNB]**

	26:22	**Base Mip Level** Project: All Format: U4.1 FormatDesc Range [0.0,13.0] Specifies which mip level is considered the “base” level when determining mag-vs-min filter and selecting the “base” mip level.

SAMPLER_STATE

21:20 Mip Mode Filter

Project: All

Format: U2 enumerated type

This field determines if and how mip map levels are chosen and/or combined when texture filtering.

Value	Name	Description	Project
0h	MIPFILTER_NONE	Disable mip mapping – force use of the mipmap level corresponding to **Min LOD**.	All
1h	MIPFILTER_NEAREST	Nearest, Select the nearest mip map	All
2h	Reserved		All
3h	MIPFILTER_LINEAR	Linearly interpolate between nearest mip maps (combined with linear min/mag filters this is analogous to "Trilinear" filtering).	All

Programming Notes

MIPFILTER_LINEAR is not supported for surface formats that do not support "Sampling Engine Filtering" as indicated in the Surface Formats table unless using the sample_c message type.

19:17 Mag Mode Filter

Project: All

Format: U2 enumerated type

This field determines how texels are sampled/filtered when a texture is being "magnified" (enlarged). For volume maps, this filter mode selection also applies to the 3rd (inter-layer) dimension.

Value	Name	Description	Project
0h	MAPFILTER_NEAREST	Sample the nearest texel	All
1h	MAPFILTER_LINEAR	Bilinearly filter the 4 nearest texels	All
2h	MAPFILTER_ANISOTROPIC	Perform an "anisotropic" filter on the chosen mip level	All
3h-5h	Reserved		All
6h	MAPFILTER_MONO	Perform a monochrome convolution filter	All
7h	Reserved		All

Programming Notes

Only MAPFILTER_NEAREST and MAPFILTER_LINEAR are supported for surfaces of type SURFTYPE_3D.

Only MAPFILTER_NEAREST is supported for surface formats that do not support

		"Sampling Engine Filtering" as indicated in the Surface Formats table unless using the sample_c message type. MAPFILTER_MONO: Only CLAMP_BORDER texture addressing mode is supported. . Both **Mag Mode Filter** and **Min Mode Filter** must be programmed to MAPFILTER_MONO. **Mip Mode Filter** must be MIPFILTER_NONE. Only valid on surfaces with **Surface Format** MONO8 and with **Surface Type** SURFTYPE_2D. MAPFILTER_ANISOTROPIC may cause artifacts at cube edges if enabled for cube maps with the TEXCOORDMODE_CUBE addressing mode. MAPFILTER_ANISOTROPIC will be overridden to MAPFILTER_LINEAR when using a sample_l or sample_l_c message type or when **Force LOD to Zero** is set in the message header. **[DevBW, DevCL] Errata: Force LOD to Zero** will not cause MAPFILTER_ANISOTROPIC to get forced to MAPFILTER_LINEAR and instead it will have to be worked around using sample_l or sample_l_c.
	16:14	**Min Mode Filter** Project: All Format: U2 enumerated type FormatDesc This field determines how texels are sampled/filtered when a texture is being "minified" (shrunk). For volume maps, this filter mode selection also applies to the 3rd (inter-layer) dimension. See **Mag Mode Filter**
	13:3	**Texture LOD Bias** Project: All Format: S4.6 2's complement FormatDesc Range [-16.0, 16.0) This field specifies the signed bias value added to the calculated texture map LOD prior to min-vs-mag determination and mip-level clamping. Assuming mipmapping is enabled, a positive LOD bias will result in a somewhat blurrier image (using less-detailed mip levels) and possibly higher performance, while a negative bias will result in a somewhat crisper image (using more-detailed mip levels) and may lower performance. **Programming Notes** There is no requirement or need to offset the LOD Bias in order to produce a correct LOD for texture filtering (as was required for correct bilinear and anisotropic filtering in some legacy devices).

SAMPLER_STATE

	2:0	**Shadow Function** Project: All Format: U3 enumerated type FormatDesc This field is used for shadow mapping support via the sample_c message type, and specifies the specific comparison operation to be used. The comparison is between the texture sample red channel (except for alpha-only formats which use the alpha channel), and the "ref" value provided in the input message. **Value** / **Name** / **Description** / **Project** 0h PREFILTEROP_ALWAYS All 1h PREFILTEROP_NEVER All 2h PREFILTEROP_LESS All 3h PREFILTEROP_EQUAL All 4h PREFILTEROP_LEQUAL All 5h PREFILTEROP_GREATER All 6h PREFILTEROP_NOTEQUAL All 7h PREFILTEROP_GEQUAL All
1	31:22	**Min LOD** Project: All Format: U4.6 in LOD units FormatDesc Range [0.0, 13.0], where the upper limit is also bounded by the **Max LOD**. This field specifies the minimum value used to clamp the computed LOD after LOD bias is applied. Note that the minification-vs.-magnification status is determined after LOD bias and before this maximum (resolution) mip clamping is applied. The integer bits of this field are used to control the "maximum" (highest resolution) mipmap level that may be accessed (where LOD 0 is the highest resolution map). The fractional bits of this value effectively clamp the inter-level trilinear blend factor when trilinear filtering is in use. **Programming Notes** If **Min LOD** is greater than **Max LOD**, **Min LOD** takes precedence, i.e. the resulting LOD will always be **Min LOD**. This field must be zero if the **Min** or **Mag Mode Filter** is set to MAPFILTER_MONO

SAMPLER_STATE

	21:12	**Max LOD** Project: All Format: U4.6 in LOD units FormatDesc Range [0.0, 13.0] This field specifies the maximum value used to clamp the computed LOD after LOD bias is applied. Note that the minification-vs.-magnification status is determined after LOD bias and before this minimum (resolution) mip clamping is applied. The integer bits of this field are used to control the "minimum" (lowest resolution) mipmap level that may be accessed. The fractional bits of this value effectively clamp the inter-level trilinear blend factor when trilinear filtering is in use. Force the mip map access to be between the mipmap specified by the integer bits of the Min LOD and the ceiling of the value specified here. **Programming Notes** If **Min LOD** is greater than **Max LOD**, **Min LOD** takes precedence, i.e. the resulting LOD will always be **Min LOD**.
	11:10	**Reserved** Project: All Format: MBZ
	9	**Cube Surface Control Mode** Project: All Format: U1 enumerated type FormatDesc When sampling from a SURFTYPE_CUBE surface, this field controls whether the **TC* Address Control Mode** fields are interpreted as programmed or overridden to TEXCOORDMODE_CUBE. **Value** **Name** **Description** **Project** 0h CUBECTRLMODE_PROGRAMMED All 1h CUBECTRLMODE_OVERRIDE All

SAMPLER_STATE

	8:6	**TCX Address Control Mode** Project: All Format: U3 enumerated type FormatDesc Controls how the 1st (TCX, aka U) component of input texture coordinates are mapped to texture map addresses – specifically, how coordinates "outside" the texture are handled (wrap/clamp/mirror). The setting of this field is subject to being overridden by the **Cube Surface Control Mode** field when sampling from a SURFTYPE_CUBE surface.

Value	Name	Description	Project
0h	TEXCOORDMODE_WRAP	Map is repeated in the U direction	All
1h	TEXCOORDMODE_MIRROR	Map is mirrored in the U direction	All
2h	TEXCOORDMODE_CLAMP	Map is clamped to the edges of the accessed map	All
3h	TEXCOORDMODE_CUBE	For cube-mapping, filtering in edges access adjacent map faces	All
4h	Reserved		All
5h	TEXCOORDMODE_MIRROR_ONCE	Map is mirrored once about origin, then clamped	All
6h-7h	Reserved		All

Programming Notes

When using cube map texture coordinates, only TEXCOORDMODE_CLAMP and TEXCOORDMODE_CUBE settings are valid, and each TC component must have the same Address Control mode.

When TEXCOORDMODE_CLAMP is used when accessing a cube map, the map's **Cube Face Enable** field must be programmed to 111111b (all faces enabled).

MAPFILTER_MONO: Texture addressing modes must all be set to TEXCOORDMODE_CLAMP_BORDER. Software must pad the border texels within the map itself with 0.

TEXCOORDMODE_MIRROR and TEXCOORDMODE_MIRROR_ONCE cannot be used with the sample_unorm* message types.

	5:3	**TCY Address Control Mode** Project: All Format: U3 enumerated type FormatDesc Controls how the 2nd (TCY, aka V) component of input texture coordinates are mapped to texture map addresses – specifically, how coordinates "outside" the texture are handled (wrap/clamp/mirror). See **Address TCX Control Mode** above for details

SAMPLER_STATE		
	2:0	**TCZ Address Control Mode** Project: All Format: U3 enumerated type FormatDesc Controls how the 3rd (TCZ) component of input texture coordinates are mapped to texture map addresses – specifically, how coordinates "outside" the texture are handled (wrap/clamp/mirror). See **Address TCX Control Mode** above for details **Programming Notes:** **[DevSNB]:** if this field is set to TEXCOORDMODE_CLAMP_BORDER samples outside the map will clamp to 0 instead of boarder color
2	31:5	**Reserved**
	4:0	**Reserved** Project: All Format: MBZ
3	31:29	**Reserved; MBZ**
	28:26	Reserved : MBZ
	25	**ChromaKey Enable** Project: All Format: Enable FormatDesc This field enables the chroma key function. **Programming Notes** Supported only on a specific subset of surface formats. See section 0 "Surface Formats" for supported formats. This field must be disabled if min or mag filter is MAPFILTER_MONO or MAPFILTER_ANISOTROPIC. This field must be disabled if used with a surface of type SURFTYPE_3D.
	24:23	**ChromaKey Index** Project: All Format: U2 FormatDesc Range [0,3] This field specifies the index of the ChromaKey Table entry associated with this Sampler. This field is a "don't care" unless **ChromaKey Enable** is ENABLED.

SAMPLER_STATE

	22	**ChromaKey Mode** Project: All Format: U1 enumerated type FormatDesc This field specifies the behavior of the device in the event of a ChromaKey match. This field is ignored if ChromaKey is disabled. KEYFILTER_KILL_ON_ANY_MATCH: In this mode, if any contributing texel matches the chroma key, the corresponding pixel mask bit for that pixel is cleared. The result of this operation is observable only if the **Killed Pixel Mask Return** flag is set on the input message. KEYFILTER_REPLACE_BLACK: In this mode, each texel that matches the chroma key is replaced with (0,0,0,0) (black with alpha=0) prior to filtering. For YCrCb surface formats, the black value is A=0, R(Cr)=0x80, G(Y)=0x10, B(Cb)=0x80. This will tend to darken/fade edges of keyed regions. Note that the pixel pipeline must be programmed to use the resulting filtered texel value to gain the intended effect, e.g., handle the case of a totally keyed-out region (filtered texel alpha==0) through use of alpha test, etc. **Value** / **Name** / **Description** / **Project** 0h / KEYFILTER_KILL_ON_ANY_MATCH / / All 1h / KEYFILTER_REPLACE_BLACK / / All
	21:19	**Maximum Anisotropy** Project: All Format: U3 enumerated type FormatDesc This field clamps the maximum value of the anisotropy ratio used by the MAPFILTER_ANISOTROPIC filter (Min or Mag Mode Filter). **Value** / **Name** / **Description** / **Project** 0h / ANISORATIO_2 / At most a 2:1 aspect ratio filter is used / All 1h / ANISORATIO_4 / At most a 4:1 aspect ratio filter is used / All 2h / ANISORATIO_6 / At most a 6:1 aspect ratio filter is used / All 3h / ANISORATIO_8 / At most a 8:1 aspect ratio filter is used / All 4h / ANISORATIO_10 / At most a 10:1 aspect ratio filter is used / All 5h / ANISORATIO_12 / At most a 12:1 aspect ratio filter is used / All 6h / ANISORATIO_14 / At most a 14:1 aspect ratio filter is used / All 7h / ANISORATIO_16 / At most a 16:1 aspect ratio filter is used / All

SAMPLER_STATE		
	18:13	**Address Rounding Enable** Project: All Format: 6-bit mask of enables FormatDesc Controls whether the U/V/R texture address is rounded or truncated before being used to select texels to sample. Each bit provides independent control of rounding on one texture address dimension (U/V/R) in either mag or min filter mode. **Value** **Name** **Description** **Project** 100000b U address mag filter All 010000b U address min filter All 001000b V address mag filter All 000100b V address min filter All 000010b R address mag filter All 000001b R address min filter All
	12:1	**Reserved** Project: All Format: MBZ
	0	**Non normalized Coordinates** Project: DevSNB+ Default Value: 0h Disable Format: Enable FormatDesc **Programming Notes** **TCX/Y/Z Address Control** Mode must be TEXCOORDMODE_CLAMP or TEXCOORDMODE_CLAMP_BORDER if enabled. Surface Type must be SURFTYPE_2D or SURFTYPE_3D.

2.11.3.2 Sampler State for sample_8x8 message

[DevSNB] This state definition is used only by the *sample_8x8* message. This state is stored as an array of up to 4 elements, each of which contains the dwords described here. The start of each element is spaced 16 dwords apart. The first element of the array is aligned to a 32-byte boundary. The index with range 0-3 that selects which element is being used is multiplied by 4 to determine the **Sampler Index** in the message descriptor.

Programming Notes:

- **IEF Filter Type** was dropped and is assumed to be Detailed filter
- **IEF Filter Size** was dropped and assumed to be 5x5.
- **IEF Bypass – If we have Y/G-channel masked then the IEF bypass should always be forced to 1.**

DWord	Bit	Description
0	31	**AVS Filter Type.** Defines the type of adaptive video scaler filter that will be enabled. 0: Adaptive 8-tap polyphase filter 1: Nearest filter
	30	Reserved : MBZ
	29	**IEF Bypass.** Causes IEF function to be bypassed, VSA will output neutral values. If Green(or Luma) channel is masked, we will always have IEF state set to bypass
	28	**IEF Filter Type** 0: Combo mode 1: Detail Filter
	27	**IEF Filter Size** 0: 3x3 1: 5x5 **Programming Notes:** • If **IEF Filter Type** is Advanced Filter, this field must be set to 5x5
	26:19	Reserved : MBZ
	18	**ChromaKey Enable.** This field enables chroma keying when accessing this particular texture map. **Programming Notes:** • For sample_8x8 instructions KEYFILTER_REPLACE_BLACK is assumed if chromakey is enabled. • For 10 bit formats only the 8 MSBs will be compared. Format = Enable
	17:16	**ChromaKey Index.** This field specifies the index of the ChromaKey Table entry associated with this Sampler. This field is a “don’t care” unless **ChromaKey Enable** is ENABLED. Format = U2 Range = [0,3]
	15:0	Reserved : MBZ

DWord	Bit	Description
1	31:5	**Sampler 8x8 State Pointer.** This field specifies the pointer to the SAMPLER_8x8_STATE structure. This pointer is relative to the **General State Base Address** for **[DevILK]** or the **Dynamic State Base Address** for **[DevSNB+]**. **Programming Notes:** • This field must be set to the same value in all sample_8x8 type SAMPLER_STATE instances applied to a given primitive. • **[DevSNB+]:** PIPE_CONTROL with **State/Instruction Cache Invalidate** set *and* the **CS Stall** field set is required between primitives that use different values of this field. • This pointer must be aligned to 512 bits. Format = DynamicStateOffset[31:5]
	4:0	Reserved : MBZ
2	31:16	Reserved : MBZ
	15:8	**Global Noise Estimation.** Global noise estimation of previous frame from DI. Format = U8 (default = 22)
	7:4	**Strong Edge Threshold.** If EM > **Strong Edge Threshold**, the basic VSA detects a strong edge. Format = U4 (default = 8)
	3:0	**Weak Edge Threshold.** If **Strong Edge Threshold** > EM > **Weak Edge Threshold**, the basic VSA detects a weak edge. Format = U4 (default = 1)
3	31	Reserved : MBZ
	30:28	**Strong Edge Weight.** Sharpening strength when a strong edge is found in basic VSA. Format = U3 (default = 7)
	27	Reserved : MBZ
	26:24	**Regular Weight.** Sharpening strength when a weak edge is found in basic VSA. Format = U3 (default = 2)
	23	Reserved : MBZ
	22:20	**Non Edge Weight.** Sharpening strength when no edge is found in basic VSA. Format = U3 (default = 1)
	19:14	**Gain Factor.** User control sharpening strength. Format = U6 (default = 40)
	13:11	Reserved : MBZ
	10:6	**R3c Coefficient.** IEF smoothing coefficient, see IEF map. Format = U0.5 (default = (59+2) >> 2)
	5	Reserved : MBZ

DWord	Bit	Description
	4:0	**R3x Coefficient.** IEF smoothing coefficient, see IEF map. Format = U0.5 (default = ((25+2) >> 2)
4	31	Reserved : MBZ
	30:26	**R5c Coefficient.** IEF smoothing coefficient, see IEF map. Format = U0.5 (default = 3)
	25	Reserved : MBZ
	24:20	**R5cx Coefficient.** IEF smoothing coefficient, see IEF map. Format = U0.5 (default = 8)
	19	Reserved : MBZ
	18:14	**R5x Coefficient.** IEF smoothing coefficient, see IEF map. Format = U0.5 (default = 9)
	13:12	Reserved : MBZ
	11:8	**Steepness Threshold.** VSA uses steepness only when greater than this threshold. Format = U4 (default = 0)
	7	**Steepness Boost.** Used to increase effect of steepness. Format = Enable (default = 0)
	6:3	**MR Threshold.** VSA uses MR only when greater than this threshold. Format = U4 (default = 5)
	2	**MR Boost.** Used to increase effect of MR. Format = Enable (default = 0)
	1:0	Reserved : MBZ
5	31:24	**PWL1 Point 4.** Point 4 for PWL of *both* sharpening and smoothing strength. Format = U8 (default = 26)
	23:16	**PWL1 Point 3.** Point 3 for PWL of *both* sharpening and smoothing strength. Format = U8 (default = 16)
	15:8	**PWL1 Point 2.** Point 2 for PWL of *both* sharpening and smoothing strength. Format = U8 (default = 12)
	7:0	**PWL1 Point 1.** Point 1 for PWL of *both* sharpening and smoothing strength. Format = U8 (default = 4)
6	31:24	**PWL1 R3 Bias 1.** Bias 1 for PWL of smoothing strength. Format = U8 (default = 98)
	23:16	**PWL1 R3 Bias 0.** Bias 0 for PWL of smoothing strength. Format = U8 (default = 127)
	15:8	**PWL1 Point 6.** Point 6 for PWL of *both* sharpening and smoothing strength. Format = U8 (default = 160)

DWord	Bit	Description
	7:0	**PWL1 Point 5.** Point 5 for PWL of *both* sharpening and smoothing strength. Format = U8 (default = 40)
7	31:24	**PWL1 R3 Bias 5.** Bias 5 for PWL of smoothing strength. Format = U8 (default = 0)
	23:16	**PWL1 R3 Bias 4.** Bias 4 for PWL of smoothing strength. Format = U8 (default = 44)
	15:8	**PWL1 R3 Bias 3.** Bias 3 for PWL of smoothing strength. Format = U8 (default = 64)
	7:0	**PWL1 R3 Bias 2.** Bias 2 for PWL of smoothing strength. Format = U8 (default = 88)
8	31:24	**PWL1 R5 Bias 2.** Bias 2 for PWL of sharpening strength. Format = U8 (default = 32)
	23:16	**PWL1 R5 Bias 1.** Bias 1 for PWL of sharpening strength. Format = U8 (default = 32)
	15:8	**PWL1 R5 Bias 0.** Bias 0 for PWL of sharpening strength. Format = U8 (default = 3)
	7:0	**PWL1 R3 Bias 6.** Bi as 6 for PWL of smoothing strength. Format = U8 (default = 0)
9	31:24	**PWL1 R5 Bias 6.** Bias 6 for PWL of sharpening strength. Format = U8 (default = 88)
	23:16	**PWL1 R5 Bias 5.** Bias 5 for PWL of sharpening strength. Format = U8 (default = 108)
	15:8	**PWL1 R5 Bias 4.** Bias 4 for PWL of sharpening strength. Format = U8 (default = 100)
	7:0	**PWL1 R5 Bias 3.** Bias 3 for PWL of sharpening strength. Format = U8 (default = 58)
10	31:24	**PWL1 R3 Slope 3.** Slope 3 for PWL of smoothing strength. Format = S3.4 2's complement (default = -32)
	23:16	**PWL1 R3 Slope 2.** Slope 2 for PWL of smoothing strength. Format = S3.4 2's complement (default = -96)
	15:8	**PWL1 R3 Slope 1.** Slope 1 for PWL of smoothing strength. Format = S3.4 2's complement (default = -20)
	7:0	**PWL1 R3 Slope 0.** Slope 0 for PWL of smoothing strength. Format = S3.4 2's complement (default = -116)

DWord	Bit	Description
11	31:24	**PWL1 R5 Slope 0.** Slope 0 for PWL of sharpening strength. Format = S3.4 2's complement (default = 116)
	23:16	**PWL1 R3 Slope 6.** Slope 6 for PWL of smoothing strength. Format = S3.4 2's complement (default = 0)
	15:8	**PWL1 R3 Slope 5.** Slope 5 for PWL of smoothing strength. Format = S3.4 2's complement (default = 0)
	7:0	**PWL1 R3 Slope 4.** Slope 4 for PWL of smoothing strength. Format = S3.4 2's complement (default = -50)
12	31:24	**PWL1 R5 Slope 4.** Slope 4 for PWL of sharpening strength. Format = S3.4 2's complement (default = 9)
	23:16	**PWL1 R5 Slope 3.** Slope 3 for PWL of sharpening strength. Format = S3.4 2's complement (default = 67)
	15:8	**PWL1 R5 Slope 2.** Slope 2 for PWL of sharpening strength. Format = S3.4 2's complement (default = 104)
	7:0	**PWL1 R5 Slope 1.** Slope 1 for PWL of sharpening strength. Format = S3.4 2's complement (default = 0)
13	31:28	**Maximum Limiter.** Strength of overshoot limiter. Format = U0.4 (default = 11)
	27:24	**Minimum Limiter.** Strength of undershoot limiter. Format = U0.4 (default = 10)
	23:20	Reserved : MBZ
	19:16	**Limiter Boost.** Used to increase limiter strength Format = U0.4 (default = 0)
	15:8	**PWL1 R5 Slope 6.** Slope 6 for PWL of sharpening strength. Format = S3.4 2's complement (default = -15)
	7:0	**PWL1 R5 Slope 5.** Slope 5 for PWL of sharpening strength. Format = S3.4 2's complement (default = -3)
14	31:18	Reserved : MBZ
	17:8	**Clip Limiter.** If extreme point is on the boundary of the neighborhood, adjust limiter's strength. Format = U10 (default = 130)
	7:0	Reserved : MBZ

2.11.3.3 For deinterlace message

This state definition is used only by the *deinterlace* message. This state is stored as an array of up to 8 elements, each of which contains the dwords described here. The start of each element is spaced 8 dwords apart. The first element of the array is aligned to a 32-byte boundary. The index with range 0-7 that selects which element is being used is multiplied by 2 to determine the **Sampler Index** in the message descriptor.

DWord	Bit	Description
0	31:24	**Denoise STAD Threshold.** Threshold for denoise sum of temporal absolute differences. Format = U8
	23:16	**Denoise Maximum History.** Maximum allowed value for denoise history. Format = U8 Range = [128,240]
	15:8	**Denoise History Delta.** Amount that denoise_history is increased. Format = U8 Range = [0,15]
	7:0	**Denoise ASD Threshold.** Threshold for denoise absolute sum of differences. Format = U8 Range = [0,63]
1	31:30	Reserved : MBZ
	29:24	**Temporal Difference Threshold.** Format = U6 **Programming Notes:** o **Temporal Difference Threshold** – **Low Temporal Difference Threshold** must be larger than or equal to 0 and less than or equal to 16.
	23:22	Reserved : MBZ

DWord	Bit	Description
	21:16	**Low Temporal Difference Threshold.** Format = U6 **Programming Notes:** o **Temporal Difference Threshold** – **Low Temporal Difference Threshold** must be larger than 0 and less than or equal to 16.
	15:13	**STMM C2:** Bias for divisor in STMM equation. Format = U3 Range = [0,7] representing values [1,8]
	12:8	**Denoise Moving Pixel Threshold.** Threshold for number of moving pixels to declare a block to be moving. Format = U5 Range = [0,16]
	7:0	**Denoise Threshold for Sum of Complexity Measure.** Format = U8
2	31:24	**Good Neighbor Threshold.** Maximum difference from current pixel for neighboring pixels to be considered a good neighbor. Format = U8 Range = [0,63]
	23:16	**Denoise Edge Threshold.** Threshold for detecting an edge in denoise. Format = U8 Range = [0,15]
	15:8	**Block Noise Estimate Edge Threshold.** Threshold for detecting an edge in block noise estimate. Format = U8 Range = [0,15]

DWord	Bit	Description
	7:0	**Block Noise Estimate Noise Threshold.** Threshold for noise maximum/minimum. Format = U8 Range = [0,31]
3	31	**STMM Blending Constant Select.** Format = U1 0: Use the blending constant for small values of STMM for stmm_md_th 1: Use the blending constant for large values of STMM for stmm_md_th
	30:24	**Blending constant across time for large values of STMM.** Format = U7
	23:16	**Blending constant across time for small values of STMM.** Format = U8
	15:14	Reserved : MBZ
	13:8	**Multiplier for VECM.** Determines the strength of the vertical edge complexity measure. Format = U6
	7:0	**Maximum STMM.** Largest allowed STMM in blending equations. Format = U8
4	31:24	**Minimum STMM.** Smallest allowed STMM in blending equations. Format = U8
	23:22	**STMM Shift Down.** Amount to shift STMM down (quantize to fewer bits). Format = U2 0: Shift by 4 1: Shift by 5 2: Shift by 6 3: Reserved

DWord	Bit	Description
	21:20	**STMM Shift Up.** Amount to shift STMM up (set range). Format = U2 0: Shift by 6 1: Shift by 7 2: Shift by 8 3: Reserved
	19:16	**STMM Output Shift.** Amount to shift output of STMM blend equation**.** **Programming Notes:** • The value of this field must satisfy the following equation: stmm_max – stmm_min = 2 ^ stmm_output_shift Format = U4 Range = [0,16]
	15:8	**SDI Threshold.** Threshold for angle detection in SDI algorithm. Format = U8
	7:0	**SDI Delta.** Delta value for angle detection in SDI algorithm. Format = U8
5	31:24	**SDI Fallback Mode 1 T1 Constant.** Format = U8
	23:16	**SDI Fallback Mode 1 T2 Constant.** Format = U8
	15:8	**SDI Fallback Mode 2 Constant (Angle2x1).** Format = U8
	7:0	**FMD Temporal Difference Threshold.** Format = U8

DWord	Bit	Description
6	31:24	**FMD #1 Vertical Difference Threshold.** Format = U8
	23:16	**FMD #2 Vertical Difference Threshold.** Format = U8
	15:14	Reserved : MBZ
	13:8	**FMD Tear Threshold.** Format = U6
	7	Reserved : MBZ
	6	**Progressive DN.** Indicates that the denoise algorithm should assume progressive input when filtering neighboring pixels. **DI Enable** must be disabled when this field is enabled. Format = Enable 0: DN assumes interlaced video and filters alternate lines together 1: DN assumes progressive video and filters neighboring lines together
	5	**DN/DI First Frame.** Indicates that this is the first frame of the stream, so previous clean is not available Format = Enable 0: Not first field; previous clean surface state is valid 1: First field; previous clean surface state is invalid
	4	**DN/DI Stream ID.** Distinguishes between the two simultaneous streams that are supported. Used to update the GNE and FMD counters for that stream. Format = U1
	3	**DN/DI Top First.** Indicates the top field is first in sequence, otherwise bottom is first Format = Enable 0 = Bottom field occurs first in sequence 1 = Top field occurs first in sequence

DWord	Bit	Description
	2	**DI Partial.** If **DI Enable** and **DI Partial** are both enabled, the deinterlacer will output the partial VDI writeback message. Format = Enable 0: Output normal VDI writeback message (only if **DI Enable** is enabled also) 1: Output partial VDI writeback message (only if **DI Enable** is enabled also)
	1	**DI Enable.** Deinterlacer is bypassed if this is disabled: the output is the same as the input (same as a 2:2 cadence). FMD and STMM are not calculated and the values in the response message are 0. Format = Enable 0: Do not calculate DI 1: Calculate DI **Programming Notes:** o **DI Enable** and **DN Enable** cannot both be disabled.
	0	**DN Enable.** Denoise is bypassed if this is low – BNE is still calculated and output, but the denoised fields are not. VDI does not read in the denoised previous frame but uses the pointer for the original previous frame. Format = Enable 0: Do not denoise frame 1: Denoise frame **Programming Notes:** o **DI Enable** and **DN Enable** cannot both be disabled.
7	31:23	**Column Width Minus1** This field specifies the (column width-1) / stride in units of blocks (Each blocks has width 16 pixels). A column width * 16 that equals the width of the frame means the walker will walk to the end of the frame. Format = U9 Range = [0, 511] representing column widths [1 to 512] (interpret value as binary value + 1)

DWord	Bit	Description
	31:19	Reserved : MBZ
	18	**VDI Walker Enable** Format = U1 0: Walker Disabled. Use XY generated by Driver. 1: Walker Enabled. Use XY generated by VDIunit. Programming Note: When enabled frame size should be aligned to 16x8 in DN only mode and 16x4 in DI enabled mode
	17:16	**FMD for 2nd field of previous frame.** Format = U2 0: Deinterlace (not progressive output) 1: Put together with previous field in sequence (1st field of previous frame). 2: Put together with next field in sequence (1st field of current frame).
	15:10	Reserved : MBZ
	9:8	**FMD for 1st field of current frame.** Format = U2 0: Deinterlace (not progressive output). 1: Put together with previous field in sequence (2nd field of previous frame). 2: Put together with next field in sequence (2nd field of current frame).
	7:0	Reserved : MBZ

2.11.4 SAMPLER_8x8_STATE [DevSNB+]

SAMPLER_8x8_STATE			
Project:	[DevSNB+]	**Length Bias:**	2

The 8x8 coefficients and other state used by the sample_8x8 message are stored as indirect state, pointed to by a field in SAMPLER_STATE. There are four different tables loaded using this structure (0X, 0Y, 1X, and 1Y). Each table is stored as an array of 17 elements, each with either 4 or 8 coefficients.

DWord	Bit	Description
0	31:24	**Table 0X Filter Coefficient[0,3]** Project: [DevSNB+] Format: S1.6 in 2's complement format Range **[DevSNB]:** Range = [0.0, +2.0) **[DevSNB+]:** Range = [-2.0, +2.0)
	23:16	**Table 0X Filter Coefficient[0,2]** Project: All Format: S1.6 in 2's complement format Range: [-1, +1)
	15:8	**Table 0X Filter Coefficient[0,1]** Project: All Format: S1.6 in 2's complement format Range $[-2^{-1}, +2^{-1})$ **Programming Notes** Must be zero if the format is R10G10B10A2_UNORM or R8G8B8A8_UNORM

SAMPLER_8x8_STATE		
	7:0	**Table 0X Filter Coefficient[0,0]** Format: S1.6 in 2's complement format Range $[-2^{-2}, +2^{-2})$ **Programming Notes** Must be zero if the format is R10G10B10A2_UNORM or R8G8B8A8_UNORM
1	31:24	**Table 0X Filter Coefficient[0,7]** Project: All Format: S1.6 FormatDesc: in 2's complement format Range $[-2^{-2}, +2^{-2})$
	23:16	**Table 0X Filter Coefficient[0,6]** Format: S1.6 FormatDesc: in 2's complement format Range $[-2^{-1}, +2^{-1})$
	15:8	**Table 0X Filter Coefficient[0,5]** Format: S1.6 in 2's complement format Range [-1, +1)
	7:0	**Table 0X Filter Coefficient[0,4]** Format: S1.6 in 2's complement format Range **[DevSNB+]:** Range = [-2.0, +2.0)
2:3		**Table 0Y Filter Coefficient[0,7:0]** This table has the same layout as Table 0X above.

SAMPLER_8x8_STATE		
4	31:24	**Table 1X Filter Coefficient[0,3]** Format: S1.6 FormatDesc; in 2's complement format Range [0.0, +2.0) BitFieldDesc
	23:16	**Table 1X Filter Coefficient[0,2]** Format: S1.6 FormatDesc Range [-1, +1) BitFieldDesc
	15:0	**Reserved** Project: All Format: MBZ
5	31:16	**Reserved** Project: All Format: MBZ
	15:8	**Table 1X Filter Coefficient[0,5]** Format: S1.6 FormatDesc: in 2's complement format Range [-1, +1) BitFieldDesc
	7:0	**Table 1X Filter Coefficient[0,4]** Format: S1.6 FormatDesc: in 2's complement format Range [0.0, +2.0) BitFieldDesc
6:7		**Table 1Y Filter Coefficient[0,7:0]** This table has the same layout as Table 1X above.
8:15		**Filter Coefficient[1,7:0]** Default Value: 0h Desc Format: OpCode
16:23	31:29	**Filter Coefficient[2,7:0]**
…		
128:135		**Filter Coefficient[16,7:0]**

SAMPLER_8x8_STATE

136	31:24	**Default Sharpness Level** Project: All Security: None Default Value: 0h DefaultVaueDesc Mask: MMIO(0x2000)#16 Format: U8 FormatDesc Address: GraphicsAddress[31:0] Surface Type: U32 Range 0..2^32-1 When adaptive scaling is off, determines the balance between sharp and smooth scalers. **Value** **Name** **Description** **Project** 0 contribute 1 from the smooth scalar 255 contribute 1 from the sharp scalar All
	23:16	**Max Derivative 4 Pixels** Format: U8 FormatDesc Used in adaptive filtering to specify the lower boundary of the smooth 4 pixel area.
	15:8	**Max Derivative 8 Pixels** Format: U8 FormatDesc Used in adaptive filtering to specify the lower boundary of the smooth 8 pixel area.
	7	**Reserved** Project: All Format: MBZ
	6:4	**Transition Area with 4 Pixels** Format: U8 FormatDesc Used in adaptive filtering to specify the width of the transition area for the 4 pixel calculation.
	3	**Reserved** Project: All Format: MBZ
	2:0	**Transition Area with 8 Pixels** Format: U3 FormatDesc Used in adaptive filtering to specify the width of the transition area for the 8 pixel calculation
137	31:23	**Reserved** Project: All Format: MBZ
	22	**Bypass X Adaptive Filtering** Format: Disable FormatDesc When disabled, the X direction will use **Default Sharpness Level** to blend between the smooth and sharp filters rather than the calculated value. **Value** **Name** **Description** **Project** 1 Disable Disable X adaptive filtering 0 Enable Enable X adaptive filtering

SAMPLER_8x8_STATE

	21	**Bypass Y Adaptive Filte** Format: Disable FormatDesc When disabled the, Y direction will use Default Sharpness Level to blend between the smooth and sharp filters rather than the calculated value. **Value** **Name** **Description** **Project** 1 Disable Disable X adaptive filtering 0 Enable Enable X adaptive filtering
	20:2	**Reserved** Project: All Format: MBZ
	1	**Adaptive Filter for all channels** Project: All Security: None Access: None Exists If: Always Default Value: 0h DefaultVaueDesc Mask: MMIO(0x2000)#16 Format: U32 FormatDesc Address: GraphicsAddress[31:0] Surface Type: U32 Range 0..2^32-1 Only to be enabled if 8-tap Adaptive filter mode is on. Else it should be disabled. **Value** **Name** **Description** **Project** 1 Enable Enable adaptive filter on UV/RB channels 0 Disable Disable adaptive filter on UV/RB channels
	0	**BitFieldName** This should be always set to 0 for YUV input and can be enabled/disabled for RGB input. This should be enabled only if we enable 8-tap adaptive filter for RGB input. **Value** **Name** **Description** **Project** 1 Enable Enable the RGB Adaptive filter using the equation (Y=(R+2G+B)>>2) 0 Disable Disable the RGB Adaptive equation and use G-Ch directly for adaptive filter

2.11.5 3DSTATE_CHROMA_KEY

3DSTATE_CHROMA_KEY			
Project:	All	**Length Bias:**	2

The 3DSTATE_CHROMA_KEY instruction is used to program texture color/chroma-key key values. A table containing four set of values is supported. The **ChromaKey Index** sampler state variable is used to select which table entry is associated with the map. Texture chromakey functions are enabled and controlled via use of the **ChromaKey Enable** texture sampler state variable.

Texture Color Key (keying on a paletted texture index) is not supported.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 04h 3DSTATE_CHROMA_KEY Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:30	**ChromaKey Table Index** Project: All Format: U2 index Range 0..3 Selects which entry in the ChromaKey table is to be loaded
	29:0	**Reserved** Project: All Format: MBZ
2	31:0	**ChromaKey Low Value** This field specifies the “low” (minimum) value of the chroma key range. Texel samples are considered “matching the key” if each component of the texel falls within the (inclusive) chroma range. See **ChromaKey High Value** for further format, programming info.

3DSTATE_CHROMA_KEY

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

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

2.11.6 3DSTATE_SAMPLER_PALETTE_LOAD0

3DSTATE_SAMPLER_PALETTE_LOAD0			
Project:	All	**Length Bias:**	2

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

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 02h 3DSTATE_SAMPLER_PALETTE_LOAD0 Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1..n	31:24	**Palette Alpha[0:N-1]** Project: [DevCTG-A+] Alpha values loaded into the first N entries of the texture palette. Format = U8
	23:0	**Palette Color[0:N-1]** Project: All Colors loaded into the first N entries of the texture color palette. Format = Bits 23:0 = U24 interpreted as RGB_888 color as follows: [23:16] Red [15:8] Green [7:0] Blue

2.11.7 3DSTATE_SAMPLER_PALETTE_LOAD1 [DevSNB]

3DSTATE_SAMPLER_PALETTE_LOAD1			
Project:	[DevSNB]	**Length Bias:**	2

The 3DSTATE_SAMPLER_PALETTE_LOAD1 instruction is used to load 32-bit values into the second texture palette. The second texture palette is used whenever a texture with a paletted format (containing "Px...[palette1]") is referenced by the sampler.

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Ch 3DSTATE_SAMPLER_PALETTE_LOAD1 Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1..n	31:0	**Palette Color[0:N-1]** Project: All Colors loaded into the first N entries of the texture color palette. Format = Bits 31:0 = U32 interpreted as ARGB_8888 color as follows: [31:24] Alpha [23:16] Red [15:8] Green [7:0] Blue

2.11.8 3DSTATE_MONOFILTER_SIZE [DevILK+]

3DSTATE_MONOFILTER_SIZE			
Project:	[DevILK+]	**Length Bias:**	2
This state specifies the size of the filter which is used when filtering in MAPFILTER_MONO mode.			

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

2.12 Messages

Restrictions:

- Use of any message to the Sampling Engine function with the **End of Thread** bit set in the message descriptor is not allowed.

2.12.1 Initiating Messages

Execution Mask

SIMD16. The 16-bit execution mask forms the valid pixel signals. This determines which pixels are sampled and results returned to the GRF registers. Samples for invalid pixels are not overwritten in the GRF. However, if LOD needs to be computed for a subspan based on the message type and MIP filter mode and at least one pixel in the subspan being valid, the sampling engine assumes that the parameters for the upper left, upper right, and lower left pixels in the subspan are valid regardless of the execution mask, as these are needed for the LOD computation.

SIMD8. The lower 8 bits of the execution mask forms the valid pixel signals. If LOD needs to be computed based on MIP filter mode and at least one pixel in the subspan being valid, the sampling engine assumes that the parameters for the upper left, upper right, and lower left pixels in the subspan are valid regardless of the execution mask, as these are needed for the LOD computation.

SIMD4x2. The lower 8 bits of the execution mask is interpreted in groups of four. If any of the high 4 bits are asserted, that sample is valid. If any of the low 4 bits are asserted, that sample is valid. The **Write Channel Mask** rather than the execution mask determines which channels are written back to the GRF.

SIMD32. The execution mask is ignored, all pixels are considered valid and all channels are returned regardless of the execution mask.

2.12.1.1 Message Descriptor

2.12.1.1.1 Message Descriptor - [DevSNB]

The following message descriptor applies to [DevSNB]. Four more bits have been added to the message descriptor.

Bit	Description
19	**Header Present:** Specifies whether the message includes a header phase. If the header is not present (this field is zero), all of the fields normally contained in the header are assumed to be 0. Format = Enable
18	Reserved : MBZ
17:16	**SIMD Mode:** Specifies the SIMD mode of the message being sent. Format = U2 0 = SIMD4x2 1 = SIMD8 2 = SIMD16 3 = SIMD32/64
15:12	**Message Type:** Specifies the type of message being sent. Format = U4 Refer to the table in section 2.12.1.3.1 for encoding details.
11:8	**Sampler Index:** Specifies the index into the sampler state table. Ignored for “ld”, “resinfo”, and “sampleinfo” type messages. Format = U4 Range = [0,15] **Programming Notes:** • for the deinterlace message, this field must be a multiple of 2 (even) • for the sample_8x8 message, this field must be a multiple of 4
7:0	**Binding Table Index:** Specifies the index into the binding table. Format = U8 Range = [0,255]

2.12.1.2 Message Header

The message header for the sampling engine is the same regardless of the message type. If the header is not present, behavior is as if the message was sent with all fields in the header set to zero (write channel masks are all enabled and offsets are zero).

DWord	Bit	Description
M0.7	31:0	**Ignored**
M0.6	31:0	**Ignored**
M0.5	31:0	Ignored
M0.4	31:0	Ignored
M0.3	31:5	Ignored
	4:0	Ignored
M0.2	31:20	Ignored
	19:18	***Ignored***
	17	Ignored
	16	Ignored
	15	**Alpha Write Channel Mask:** Enables the alpha channel to be written back to the originating thread. 0: Alpha channel will be written back 1: Alpha channel will not be written back **Programming Notes:** • a message with all four channels masked is not allowed.. • this field is ignored for the deinterlace message. • this field must be set to zero for sample_8x8 in VSA mode.
	14	**Blue Write Channel Mask:** See Alpha Write Channel Mask
	13	**Green Write Channel Mask:** See Alpha Write Channel Mask
	12	**Red Write Channel Mask:** See Alpha Write Channel Mask
	11:8	**U Offset:** the u offset from the _aoffimmi modifier on the "sample" or "ld" instruction in DX10. Must be zero if the **Surface Type** is SURFTYPE_CUBE or SURFTYPE_BUFFER. Must be set to zero if _aoffimmi is not specified. Format is S3 2's complement. **Programming Note:** • this field is ignored for the sample_unorm*, sample_8x8, and deinterlace messages • this field is ignored if the "offu" parameter is included in the gather4* messages

DWord	Bit	Description
	7:4	**V Offset:** the v offset from the _aoffimmi modifier on the "sample" or "ld" instruction in DX10. Must be zero if the **Surface Type** is SURFTYPE_CUBE or SURFTYPE_BUFFER. Must be set to zero if _aoffimmi is not specified. Format is S3 2's complement. **Programming Note:** • this field is ignored for the sample_unorm*, sample_8x8, and deinterlace messages • this field is ignored if the "offu" parameter is included in the gather4* messages
	3:0	**R Offset:** the r offset from the _aoffimmi modifier on the "sample" or "ld" instruction in DX10. Must be zero if the **Surface Type** is SURFTYPE_CUBE or SURFTYPE_BUFFER. Must be set to zero if _aoffimmi is not specified. Format is S3 2's complement. **Programming Note:** • this field is ignored for the sample_unorm*, sample_8x8, and deinterlace messages
M0.1	31:0	Ignored
M0.0	31:0	Ignored

2.12.1.3 Payload Parameter Definition

The message type field in the message descriptor in combination with the message length determines which message is being sent. The table defines all of the *parameters* sent for each message type. The position of the parameters in the payload is given in the section following corresponding to the *SIMD mode* given in the table. The instruction column indicates the DX10 shader instructions expected to be translated to each message type.

All parameters are of type IEEE_Float, except those in the ld and resinfo instruction message types, which are of type S31. Any parameter indicated with a blank entry in the table is unused. A message register containing only unused parameters not included as part of the message. The response lengths given below assume all channels are unmasked. SIMD16 messages with masked channels will have reduced response length.

2.12.1.3.1 Payload Parameter Definition [DevSNB]

The table below shows all of the message types supported by the sampling engine. The **Message Type** field in the message descriptor determines which message is being sent. The **SIMD Mode** field determines the number of instances (i.e. pixels) and the formatting of the initiating and writeback messages. The **Header Present** field determines whether a header is delivered as the first phase of the message or the default header from R0 of the thread's dispatch is used. The **Message Length** field is used to vary the number of parameters sent with each message. Higher-numbered parameters are optional, and default to a value of 0 if not sent but needed for the surface being sampled.

The message lengths are computed as follows, where "N" is the number of parameters ("N" is rounded up to the next multiple of 4 for SIMD4x2), and "H" is 1 if the header is present, 0 otherwise. The maximum message length allowed to the sampler is 11. This would disallow sample_d, sample_b_c, and sample_l_c with a SIMD Mode of SIMD16.

SIMD Mode	Message Length
SIMD4x2	H + (N/4)
SIMD8	H + N
SIMD16	H + (2*N)

The response lengths are computed as follows:

	SIMD Mode	Response Length
SIMD8	SIMD4x2	1
	sample+killpix	5
	all other message types	4
	SIMD16	8 *

* For SIMD16, phases in the response length are reduced by 2 for each channel that is masked.

SIMD16 messages with six or more parameters exceed the maximum message length allowed, in which case they are not supported. This includes some forms of sample_d, sample_d_c, and sample_l_c message types.

SIMD4x2, SIMD8, and SIMD16 Messages:

Message Type	mnemonic	parameters									
		0	1	2	3	4	5	6	7	8	9
0000	sample	u	v	r	ai						
0001	sample_b	u	v	r	ai	bias					
0010	sample_l	u	v	r	ai	lod					
0011	sample_c	u	v	r	ai	ref					
0100	sample_d	u	v	r	ai	dudx	dudy	dvdx	dvdy	drdx	drdy
0101	sample_b_c	u	v	r	ai	ref	bias				
0110	sample_l_c	u	v	r	ai	ref	lod				
0111	ld	u	v	r	lod	si					
1000*	load4	u	v	r	ai						
1001*	LOD	u	v	r	ai						
1010	resinfo	lod									
1011*	sampleinfo										
1100	sample+killpix	u	v	r							

* These messages are supported only on **[DevSNB+]**.

For the SIMD32/SIMD64 messages, the input message is not defined in terms of parameters. “H” is 1 if the header is present, 0 otherwise.

[DevSNB+] SIMD32/SIMD64 Messages:

Message Type	mnemonic	Payload Layout	Message Length	Response Length
00000	sample_unorm	Pixel Shader	H + 1	8 **
00010	sample_unorm+killpix	Pixel Shader	H + 1	9 **
00011	sample_8x8	Pixel Shader	H + 1	16 *
01000	deinterlace	Pixel Shader	H + 1	†
01100	sample_unorm	Media	H + 1	8 **
01010	sample_unorm+killpix	Media	H + 1	9 **
01011	sample_8x8	Media	H + 1	16 *

* For sample_8x8, phases in the response length are reduced by 4 for each channel that is masked.

** For sample_unorm, phases in the response length are reduced by 2 for each channel that is masked.

† For deinterlace, response length depending on certain state fields. Refer to writeback message definition for details.

2.12.1.4 Message Types

The behavior of each message type is as follows:

Message Type	Description
sample	The surface is sampled using the indicated sampler state. LOD is computed using gradients between adjacent pixels. One, two, or three parameters may be specified depending on how many coordinate dimensions the indicated surface type uses. Extra parameters specified are ignored. Missing parameters are defaulted to 0. **Restriction:** if sample from a multisampled surface (Number of Multisamples is MULTISAMPLECOUNT_4), fraction of U*width has to be 0.5. same for V* height **Programming Notes:** • The Surface Type of the associated surface must be SURFTYPE_1D, SURFTYPE_2D, SURFTYPE_3D, or SURFTYPE_CUBE. • The Surface Format of the associated surface cannot be MONO8 or any UINT or SINT format. • sample is not supported in SIMD4x2 mode.

Message Type	Description
sample+killpix	The surface is sampled as in the sample message type. An additional register is returned after the sample results which contains the kill pixel mask. This message type is required to allow the result of a chroma key enabled sampler in KEYFILTER_KILL_ON_ANY_MATCH mode to affect the final pixel mask. **Programming Notes:** • The Surface Type of the associated surface must be SURFTYPE_1D, SURFTYPE_2D, SURFTYPE_3D, or SURFTYPE_CUBE. • The Surface Format of the associated surface cannot be MONO8 or any UINT or SINT format. • sample+killpix is supported only in SIMD8 mode.
sample_b	The surface is sampled using the indicated sampler state. LOD is computed using gradients between adjacent pixels, then the value in the parameter is added to the LOD for each pixel. The LOD bias delivered in the bias parameter is restricted to a range of [-16.0, +16.0). Values outside this range produce undefined results. **Programming Notes:** • The Surface Type of the associated surface must be SURFTYPE_1D, SURFTYPE_2D, SURFTYPE_3D, or SURFTYPE_CUBE. • The Surface Format of the associated surface cannot be MONO8 or any UINT or SINT format. • sample_b is not supported in SIMD4x2 mode.
sample_l	The surface is sampled using the indicated sampler state. LOD is not computed, but instead is taken from the lod parameter. **Programming Notes:** • The Surface Type of the associated surface must be SURFTYPE_1D, SURFTYPE_2D, SURFTYPE_3D, or SURFTYPE_CUBE. • The Surface Format of the associated surface cannot be a UINT or SINT format.

Message Type	Description
sample_c	The surface is sampled using the indicated sampler state. All four coordinates must be specified, however v and r may not be used depending on the indicated surface type. The ai parameter indicates the array index for a cube surface. The ref parameter specifies the reference value that is compared against the red channel of the sampled surface, and the texel is replaced with either white or black depending on the result of the comparison. The WGF sample_c_lz instruction is implemented by issuing the sample_c message with Force LOD to Zero enabled in the message header or by issuing the sample_l_c message with the LOD parameter set to zero. **Programming Notes:** • The Surface Type of the associated surface must be SURFTYPE_1D, SURFTYPE_2D, or SURFTYPE_CUBE. • The Surface Format of the associated surface must be indicated as supporting shadow mapping as indicated in the surface format table. • With sample_c, MIPFILTER_LINEAR, MAPFILTER_LINEAR, MAPFILTER_ANISOTROPIC are allowed even for surface formats that are listed as not supporting filtering in the surface formats table. • Use of the SIMD4x2 form of sample_c without Force LOD to Zero enabled in the message header is not allowed, as it is not possible for the hardware to compute LOD for SIMD4x2 messages. For [DevILK+], sample_c is not supported in SIMD4x2 mode. • Use of sample_c with SURFTYPE_CUBE surfaces is undefined with the following surface formats: I24X8_UNORM, L24X8_UNORM, A24X8_UNORM, I32_FLOAT, L32_FLOAT, A32_FLOAT. • [DevBW, DevCL] Errata: When sample_c is used on a texture map with A16_FLOAT surface format, any value read in from the texture map that is a NaN will be treated like a + inf. • [Pre-DevILK] Errata: When either the reference value or the source value from the texture map is NaN the compare value will be incorrectly replaced with 1.0 rather than 0.0 for Shadow Function of GEQUAL, GREATER, LEQUAL, or LESS.
sample_b_c	This is a combination of sample_b and sample_c. Both the LOD bias and reference values are delivered. All restrictions applying to both sample_b and sample_c must be honored.
sample_l_c	This is a combination of sample_l and sample_c. Both the LOD and reference values are delivered. All restrictions applying to both sample_l and sample_c must be honored. However, unlike sample_c, sample_l_c is allowed as a SIMD4x2 message.
sample_g *sample_d*	The surface is sampled using the indicated sampler state. LOD is computed using the gradients present in the message. The r coordinate and its gradients are required only for surface types that use the third coordinate. Usage of this message type on cube surfaces assumes that the u, v, and gradients have already been transformed onto the appropriate face, but still in [-1,+1] range. The r coordinate contains the faceid, and the r gradients are ignored by hardware. **Programming Notes:** • The Surface Type of the associated surface must be SURFTYPE_1D, SURFTYPE_2D, SURFTYPE_3D, or SURFTYPE_CUBE. • The Surface Format of the associated surface cannot be MONO8 or any UINT or SINT format.

Message Type	Description
sample_g_c *sample_d_c*	This is a combination of sample_g and sample_c. Both the gradients for calculating LOD and reference values are delivered. All restrictions applying to both sample_g and sample_c must be honored. However, unlike sample_c, sample_g_c is allowed as a SIMD4x2 message.
resinfo	The surface indicated in the surface state is not sampled. Instead, the width, height, depth, and MIP count of the surface are returned as indicated in the table below. The format of the returned data is FLOAT32 for [Pre-DevCTG], and UINT32 for [DevCTG+]. The width, height, and depth may be shifted right, per pixel, by the LOD value provided in the lod parameter to give the dimensions of the specified mip level. The lod parameter is an unsigned 32-bit integer in this mode (note that sending a signed 32-bit integer always has the same effect, as negative values are out-of-range when interpreted as unsigned integers). The Sampler State Pointer and Sampler Index are ignored.

surface type	red	green	blue	alpha
SURFTYPE_1D	(Width+1)>>LOD	[DevSNB]:Depth==0 ? 0 : Depth+1	0	MIPCount
SURFTYPE_2D	(Width+1)>>LOD	(Height+1)>>LOD	[DevSNB]:Depth==0 ? 0 : Depth+1	MIPCount
SURFTYPE_3D	(Width+1)>>LOD	(Height+1)>>LOD	(Depth+1)>>LOD	MIPCount
SURFTYPE_CUBE	(Wdith+1)>>LOD	(Height+1)>>LOD	[DevSNB+]: Depth==0 ? 0 : Depth+1	MIPCount

Message Type	Description
ld *ld2dms* *ld_mcs* *ld2dss*	The surface is sampled using a default sampler state, indicated below. The parameter contains the LOD of the mip map to be sampled. The parameter contains the sample index, which is clamped to the number of samples on the surface (supported on [DevSNB+] only). The v and r channel may be ignored depending on the indicated surface type. All incoming values are unsigned 32-bit integers in this mode. The u, v, and r parameters contain integer texel addresses on the LOD indicated in the parameter. The Sampler State Pointer and Sampler Index are ignored. For these message types, the sampler state is defaulted as follows: • min, mag, and mip filter modes are "nearest" • all address control modes are "zero" (a special mode in which any texel off the map or outside the MIP range of the surface has a value of zero in all channels, except for surface formats without an alpha channel, which will return a value of one in the alpha channel) **Programming Notes:** • The Surface Type of the associated surface must be SURFTYPE_1D, SURFTYPE_2D, SURFTYPE_3D, or SURFTYPE_BUFFER for the ld message. • The Surface Type of the associated surface must be SURFTYPE_2D for the ld_mcs , ld2dms , and ld2dss messages. • The Surface Format of the associated surface cannot be MONO8.
sampleinfo	**[DevSNB+] only:** The surface indicated in the surface state is not sampled. Instead, the number of samples (UINT32) and the sample position palette index (UINT32) for the surface are returned in the red and alpha channels respectively as UINT32 values. The sample position palette index returned in alpha is incremented by one from its value in the surface state. The Sampler State Pointer and Sampler Index are ignored. **[DevSNB] : Errata:** If the Surface Type is SURFTYPE_NULL, the values of the above state fields from SURFACE_STATE are returned, rather than zeros that would normally be expected.
LOD	**[DevSNB+] only:** The surface indicated in the surface state is not sampled. Instead, LOD is computed as if the surface will be sampled, using the indicated sampler state, and the clamped and unclamped LOD values are returned in the red and green channels, respectively, in FLOAT32 format. The blue and alpha channels are undefined, and can be masked to avoid returning them. LOD is computed using gradients between adjacent pixels. Three parameters are always specified, with extra parameters not needed for the surface being ignored. **Programming Notes:** • The Surface Type of the associated surface must be SURFTYPE_1D, SURFTYPE_2D, SURFTYPE_3D, or SURFTYPE_CUBE. • The Surface Format of the associated surface cannot be MONO8 or any UINT or SINT format. • LOD is not supported in SIMD4x2 mode. • **[DevSNB-A0, DevSNB-B0] Errata**: Fractional Bit may be off for the clamped LOD.

<table>
<tr><th>Message Type</th><th>Description</th></tr>
<tr><td>gather4

gather4_po

(load4)</td><td>

[DevSNB+] only: The surface is sampled using bilinear filtering, regardless of the filtering mode specified in the sampler state. For SURFTYPE_2D LOD is forced to zero before sampling. The samples are not filtered, but instead the four samples are returned directly in the sample's corresponding four channels as follows:

<table>
<tr><td>upper left sample = alpha channel</td><td>upper right sample = blue channel</td></tr>
<tr><td>lower left sample = red channel</td><td>lower right sample = green channel</td></tr>
</table>

Two or three parameters may be specified depending on how many coordinate dimensions the indicated surface type uses. Extra parameters specified are ignored. Missing parameters default to 0.

Programming Notes:

- The Surface Type of the associated surface must be SURFTYPE_2D or SURFTYPE_CUBE. If the message type is gather4_po, only SURFTYPE_2D is allowed.
- The Surface Format of the associated surface cannot be MONO8 or any UINT or SINT format,
- [DevSNB]: The Surface Format must be a format that consists of a single channel (i.e. red or alpha only).
- Mip Mode Filter must be set to MIPFILTER_NONE
- [DevSNB] errata: When gather4 is used with an Address Control Mode of MIRROR or MIRROR_ONCE, the odd instances of the surface will return texels in incorrect positions.

</td></tr>
</table>

Message Type	Description
sample_unorm	The surface is sampled using the indicated sampler state. 32 contiguous pixels in a 8-wide by 4-high arrangement are sampled. The U and V addresses for the upper left pixel is delivered in this message along with a Delta U and Delta V parameter. Given a pixel at (x,y) relative to the upper left pixel (where (0,0) is the upper left pixel), the U and V for that pixel are computed as follows: U(x,y) = U(0,0) + DeltaU * x V(x,y) = V(0,0) + DeltaV * y **Programming Notes:** • The Surface Type of the associated surface must be SURFTYPE_2D • The Surface Format of the associated surface must be UNORM with <= 8 bits per channel • The MIP Count, Depth, Surface Min LOD, and Min Array Element of the associated surface must be 0 • The Min and Mag Mode Filter must be MAPFILTER_NEAREST or MAPFILTER_LINEAR • The Mip Mode Filter must be MIPFILTER_NONE • The TCX and TCY Address Control Mode cannot be TEXCOORDMODE_CLAMP_BORDER TEXCOORDMODE_MIRROR TEXCOORDMODE_MIRROR_ONCE • DeltaU * Width of the associated surface must be less than or equal to 3.0 • DeltaV * Height of the associated surface must be less than or equal to 3.0
sample_unorm_RG	**[DevCTG] to [DevILK] only:** This message is identical to the sample_unorm message except it only returns the red and green channels in the writeback message. All restrictions of the sample_unorm message apply to this message also.
sample_unorm_RG *+killpix*	**[DevCTG] to [DevILK] only:** This message is identical to the sample_unorm_RG message except it returns a kill pixel mask in addition to the red and green channels in the writeback message. This message type is required to allow the result of a chroma key enabled sampler in KEYFILTER_KILL_ON_ANY_MATCH mode to affect the final pixel mask. All restrictions of the sample_unorm message apply to this message also.
sample_unorm *+killpix*	**[DevSNB+] only:** This message is identical to the sample_unorm message except it returns a kill pixel mask in addition to the selected channels in the writeback message. This message type is required to allow the result of a chroma key enabled sampler in KEYFILTER_KILL_ON_ANY_MATCH mode to affect the final pixel mask. All restrictions of the sample_unorm message apply to this message also.

Message Type	Description
sample_8x8	**[DevILK+] only:** The surface is sampled using an optional 8x8 filter followed by an optional image enhancement filter, using state defined in SAMPLER_STATE and SAMPLER_8x8_STATE. The input consists of 64 contiguous pixels in an 16-wide by 4-high arrangement. The address control mode behaves as clamp mode. The U and V addresses for the upper left pixel are delivered in this message along with a Delta U and Delta V parameter. Given a pixel at (x,y) relative to the upper left pixel (where (0,0) is the upper left pixel), the U and V for that pixel are computed as follows: U(x,y) = U(0,0) + DeltaU * x + U_2nd_Derivative * x * (x - 1)/2 V(x,y) = V(0,0) + DeltaV * y **Programming Notes:** • The Surface Type of the associated surface must be SURFTYPE_2D • The Surface Format of the associated surface must be UNORM with <= 10 bits per channel • DeltaV * Height of the associated surface must be less than 16.0 • Map Width must be >= 4 • DeltaU * Width of the associated surfaces must be less than 16.0 and greater than or equal to 0.0 • The following must be true: (DeltaU * Width / 18) <= U_2ndDerivative * Width < (64 – 2 * DeltaU * Width) / 35 • **[DevILK-A]:** If sample_8x8 or deinterlace messages are used in a thread, software must ensure that the same thread or other threads that can concurrently be running do not use any other sampling engine messages. •
deinterlace	**[DevSNB]:** The surface is deinterlaced and/or denoised, using state defined in SAMPLER_STATE. The U and V addresses for the upper left pixel are delivered in this message. **Programming Notes:** • **[DevILK-A]:** If sample_8x8 or deinterlace messages are used in a thread, software must ensure that the same thread or other threads that can concurrently be running do not use any other sampling engine messages.

Programming Notes:

- For surfaces of type SURFTYPE_CUBE, the sampling engine requires u, v, and r parameters that have already been divided by the absolute value of the parameter (u, v, or r) with the largest absolute value.

2.12.1.5 Parameter Types

sample*, LOD, and gather4 messages

For all of the sample*, LOD, and gather4 message types, all parameters are 32-bit floating point, except the 'mcs', 'offu', and 'offv' parameters. Usage of the u, v, and r parameters is as follows based on **Surface Type**. Normalized values range from [0,1] across the surface, with values outside the surface behaving as specified by the **Address Control Mode** in that dimension. Unnormalized values range from [0,n-1] across the surface, where n is the size of the surface in that dimension, with values outside the surface being clamped to the surface.

Surface Type	u	v	r	ai
SURFTYPE_1D	*normalized 'x' coordinate*	*unnormalized array index*	*ignored*	*ignored*
SURFTYPE_2D	*normalized 'x' coordinate*	*normalized 'y' coordinate*	*unnormalized array index*	*ignored*
SURFTYPE_3D	*normalized 'x' coordinate*	*normalized 'y' coordinate*	*normalized 'z' coordinate*	*ignored*
SURFTYPE_CUBE	*normalized 'x' coordinate*	*normalized 'y' coordinate*	*normalized 'z' coordinate*	*unnormalized array index*

Ld* messages

For the ld message types, all parameters are 32-bit signed integers, except the 'mcs' parameter. Usage of the u, v, and r parameters is as follows based on **Surface Type**. Unnormalized values range from [0,n-1] across the surface, where n is the size of the surface in that dimension. Input of any value outside of the range returns zero.

Surface Type	u	v	r
SURFTYPE_1D	*unnormalized 'x' coordinate*	*unnormalized array index*	*ignored*
SURFTYPE_2D	*unnormalized 'x' coordinate*	*unnormalized 'y' coordinate*	*unnormalized array index*
SURFTYPE_3D	*unnormalized 'x' coordinate*	*unnormalized 'y' coordinate*	*unnormalized 'z' coordinate*
SURFTYPE_BUFFER	*unnormalized 'x' coordinate*	*ignored*	*ignored*

2.12.1.6 SIMD16 Payload

The payload of a SIMD16 message provides addresses for the sampling engine to process 16 entities (examples of an entity are vertex and pixel). The number of parameters required to sample the surface depends on the state that the sampler/surface is in. Each parameter takes two message registers, with 8 entities, each a 32-bit floating point value, being placed in each register. Each parameter always takes a consistent position in the input payload. The length field can be used to send a shorter message, but intermediate parameters cannot be skipped as there is no way to signal this. For example, a 2D map using "sample_b" needs only u, v, and bias, but must send the r parameter as well.

DWord	Bit	Description
M1.7	31:0	**Subspan 1, Pixel 3 (lower right) Parameter 0** Specifies the value of the pixel's parameter 0. The actual parameter that maps to parameter 0 is given in the table in section 2.12.1.3. Format = IEEE Float for all sample* message types, U32 for ld and resinfo message types.
M1.6	31:0	**Subspan 1, Pixel 2 (lower left) Parameter 0**
M1.5	31:0	**Subspan 1, Pixel 1 (upper right) Parameter 0**
M1.4	31:0	**Subspan 1, Pixel 0 (upper left) Parameter 0**
M1.3	31:0	**Subspan 0, Pixel 3 (lower right) Parameter 0**
M1.2	31:0	**Subspan 0, Pixel 2 (lower left) Parameter 0**
M1.1	31:0	**Subspan 0, Pixel 1 (upper right) Parameter 0**
M1.0	31:0	**Subspan 0, Pixel 0 (upper left) Parameter 0**
M2.7	31:0	**Subspan 3, Pixel 3 (lower right) Parameter 0**
M2.6	31:0	**Subspan 3, Pixel 2 (lower left) Parameter 0**
M2.5	31:0	**Subspan 3, Pixel 1 (upper right) Parameter 0**
M2.4	31:0	**Subspan 3, Pixel 0 (upper left) Parameter 0**
M2.3	31:0	**Subspan 2, Pixel 3 (lower right) Parameter 0**
M2.2	31:0	**Subspan 2, Pixel 2 (lower left) Parameter 0**
M2.1	31:0	**Subspan 2, Pixel 1 (upper right) Parameter 0**
M2.0	31:0	**Subspan 2, Pixel 0 (upper left) Parameter 0**
M3 – Mn		Repeat packets 1 and 2 to cover all required parameters

2.12.1.7 SIMD8 Payload

This message is intended to be used in a SIMD8 thread, or in pairs from a SIMD16 thread. Each message contains sample requests for just 8 pixels.

DWord	Bit	Description
M1.7	31:0	**Subspan 1, Pixel 3 (lower right) Parameter 0** Specifies the value of the pixel's parameter 0. The actual parameter that maps to parameter 0 is given in the table in section 2.12.1.3. Format = IEEE Float for all sample* message types, U32 for ld and resinfo message types.
M1.6	31:0	**Subspan 1, Pixel 2 (lower left) Parameter 0**
M1.5	31:0	**Subspan 1, Pixel 1 (upper right) Parameter 0**
M1.4	31:0	**Subspan 1, Pixel 0 (upper left) Parameter 0**
M1.3	31:0	**Subspan 0, Pixel 3 (lower right) Parameter 0**
M1.2	31:0	**Subspan 0, Pixel 2 (lower left) Parameter 0**
M1.1	31:0	**Subspan 0, Pixel 1 (upper right) Parameter 0**
M1.0	31:0	**Subspan 0, Pixel 0 (upper left) Parameter 0**
M2 – Mn		Repeat packet 1 to cover all required parameters

2.12.1.8 SIMD4x2 Payload

DWord	Bit	Description
M1.7	31:0	**Sample 1 Parameter 3** Specifies the value of the pixel's parameter 3. The actual parameter that maps to parameter 3 is given in the table in section 2.12.1.3. Format = IEEE Float for all sample* message types, U32 for ld and resinfo message types.
M1.6	31:0	**Sample 1 Parameter 2**
M1.5	31:0	**Sample 1 Parameter 1**
M1.4	31:0	**Sample 1 Parameter 0**
M1.3	31:0	**Sample 0 Parameter 3**
M1.2	31:0	**Sample 0 Parameter 2**
M1.1	31:0	**Sample 0 Parameter 1**

DWord	Bit	Description
M1.0	31:0	**Sample 0 Parameter 0**
M2		Parameters 4-7 if present
M3		Parameters 8-11 if present

2.12.1.9 SIMD32/64 Payload

2.12.1.9.1 Pixel Shader

This position of **Delta U/V** in the pixel shader payload layout is to allow the register delivered in the pixel shader dispatch containing the coefficients for the texture coordinates to be left in their original position (Delta U = Cxs, Delta V = Cyt). The values for U and V are computed in the pixel shader into the unused positions in this register.

DWord	Bit	Description
M1.7	31:0	Ignored
M1.6	31:0	**Pixel 0 V Address** Format: sample_unorm* and sample_8x8: IEEE_Float in normalized space deinterlace: U32 (Range: [0,2046])
M1.5	31:0	**Delta V**: defines the difference in V for adjacent pixels in the Y direction. **Programming Notes**: • **Delta V** multiplied by **Height** in SURFACE_STATE must be less than or equal to 3 for sample_unorm* message types. • **Delta V** multiplied by **Height** in SURFACE_STATE must be less than 16 for the sample_8x8 message type. • This field is ignored for the deinterlace message type. Format = IEEE_Float in normalized space
M1.4	31:0	Ignored
M1.3	31:0	Ignored
M1.2	31:0	**Pixel 0 U Address** Format: sample_unorm* and sample_8x8: IEEE_Float in normalized space deinterlace: U32 (Range: [0,4095])

DWord	Bit	Description
M1.1	31:0	**U 2nd Derivative** Defines the change in the delta U for adjacent pixels in the X direction. **Programming Notes**: • This field is ignored for messages other than sample_8x8. Format = IEEE_Float in normalized space
M1.0	31:0	**Delta U**: defines the difference in U for adjacent pixels in the X direction. **Programming Notes**: • **Delta U** multiplied by **Width** in SURFACE_STATE must be less than or equal to 3 for sample_unorm* message types. • This field is ignored for the deinterlace message type. Format = IEEE_Float in normalized space

2.12.1.9.2 Media [DevILK+ only]

The position of **Delta U** and **U 2nd Derivative** in the media payload layout is intended to make media kernels more efficient. Sending a message using the media payload layout behaves identically to the pixel shader payload layout other than the position of these two fields.

DWord	Bit	Description
M1.7	31:0	**Ignored**
M1.6	31:0	**Pixel 0 V Address** Format: sample_unorm* and sample_8x8: IEEE_Float in normalized space deinterlace: U32 (Range: [0,2046])
M1.5	31:0	**Delta V**: defines the difference in V for adjacent pixels in the Y direction. **Programming Notes**: • **Delta V** multiplied by **Height** in SURFACE_STATE must be less than or equal to 3 for sample_unorm* message types. • **Delta V** multiplied by **Height** in SURFACE_STATE must be less than 16 for the sample_8x8 message type. • This field is ignored for the deinterlace message type. Format = IEEE_Float in normalized space
M1.4	31:0	Ignored
M1.3	31:0	Ignored

DWord	Bit	Description
M1.2	31:0	**Pixel 0 U Address** Format: sample_unorm* and sample_8x8: IEEE_Float in normalized space deinterlace: U32 (Range: [0,4095])
M1.1	31:0	**Delta U**: defines the difference in U for adjacent pixels in the X direction. **Programming Notes**: • **Delta U** multiplied by **Width** in SURFACE_STATE must be less than or equal to 3 for sample_unorm* message types. • This field is ignored for the deinterlace message type. Format = IEEE_Float in normalized space
M1.0	31:0	**U 2nd Derivative** Defines the change in the delta U for adjacent pixels in the X direction. **Programming Notes**: • This field is ignored for messages other than sample_8x8. Format = IEEE_Float in normalized space

2.12.2 Writeback Message

Corresponding to the four input message definitions are four writeback messages. Each input message generates a corresponding writeback message of the same type (SIMD16, SIMD8, SIMD4x2, or SIMD32/64).

2.12.2.1 SIMD16

A SIMD16 writeback message consists of up to 8 destination registers. Which registers are returned is determined by the write channel mask received in the corresponding input message. Each asserted write channel mask results in both destination registers of the corresponding channel being skipped in the writeback message, and all channels with higher numbered registers being dropped down to fill in the space occupied by the masked channel. For example, if only red and alpha are enabled, red is sent to regid+0 and regid+1, and alpha to regid+2 and regid+3. The pixels written within each destination register is determined by the execution mask on the “send” instruction.

DWord	Bit	Description
W0.7	31:0	**Subspan 1, Pixel 3 (lower right) Red:** Specifies the value of the pixel’s red channel. Format = IEEE Float, S31 signed 2’s comp integer, or U32 unsigned integer. Format depends on the **Data Return Format** programmed for the surface being sampled.
W0.6	31:0	**Subspan 1, Pixel 2 (lower left) Red**
W0.5	31:0	**Subspan 1, Pixel 1 (upper right) Red**

DWord	Bit	Description
W0.4	31:0	**Supspan 1, Pixel 0 (upper left) Red**
W0.3	31:0	**Subspan 0, Pixel 3 (lower right) Red**
W0.2	31:0	**Subspan 0, Pixel 2 (lower left) Red**
W0.1	31:0	**Subspan 0, Pixel 1 (upper right) Red**
W0.0	31:0	**Supspan 0, Pixel 0 (upper left) Red**
W1.7	31:0	**Subspan 3, Pixel 3 (lower right) Red**
W1.6	31:0	**Subspan 3, Pixel 2 (lower left) Red**
W1.5	31:0	**Subspan 3, Pixel 1 (upper right) Red**
W1.4	31:0	**Supspan 3, Pixel 0 (upper left) Red**
W1.3	31:0	**Subspan 2, Pixel 3 (lower right) Red**
W1.2	31:0	**Subspan 2, Pixel 2 (lower left) Red**
W1.1	31:0	**Subspan 2, Pixel 1 (upper right) Red**
W1.0	31:0	**Supspan 2, Pixel 0 (upper left) Red**
W2		**Subspans 1 and 0 of Green:** See W0 definition for pixel locations
W3		**Subspans 3 and 2 of Green:** See W1 definition for pixel locations
W4		**Subspans 1 and 0 of Blue:** See W0 definition for pixel locations
W5		**Subspans 3 and 2 of Blue:** See W1 definition for pixel locations
W6		**Subspans 1 and 0 of Alpha:** See W0 definition for pixel locations
W7		**Subspans 3 and 2 of Alpha:** See W1 definition for pixel locations

2.12.2.2 SIMD8

This writeback message consists of four registers, or five in the case of sample+killpix. As opposed to the SIMD16 writeback message, channels that are masked in the write channel mask are not skipped, all four channels are always returned. The masked channels, however, are not overwritten in the destination register.

For the sample+killpix message types, an additional register (W4) is included after the last channel register.

DWord	Bit	Description
W0.7	31:0	**Subspan 1, Pixel 3 (lower right) Red:** Specifies the value of the pixel's red channel. Format = IEEE Float, S31 signed 2's comp integer, or U32 unsigned integer. Format depends on the **Data Return Format** programmed for the surface being sampled.
W0.6	31:0	**Subspan 1, Pixel 2 (lower left) Red**
W0.5	31:0	**Subspan 1, Pixel 1 (upper right) Red**
W0.4	31:0	**Supspan 1, Pixel 0 (upper left) Red**
W0.3	31:0	**Subspan 0, Pixel 3 (lower right) Red**
W0.2	31:0	**Subspan 0, Pixel 2 (lower left) Red**
W0.1	31:0	**Subspan 0, Pixel 1 (upper right) Red**
W0.0	31:0	**Supspan 0, Pixel 0 (upper left) Red**
W1		**Subspans 1 and 0 of Green:** See W0 definition for pixel locations
W2		**Subspans 1 and 0 of Blue:** See W0 definition for pixel locations
W3		**Subspans 1 and 0 of Alpha:** See W0 definition for pixel locations
W4.7:1		Reserved (not written) **:** W4 is only delivered for the sample+killpix message type
W4.0	31:16	**Dispatch Pixel Mask:** This field is always 0xffff to allow dword-based ANDing with the R0 header in the pixel shader thread.
	15:0	**Active Pixel Mask:** This field has the bit for all pixels set to 1 except those pixels that have been killed as a result of chroma key with kill pixel mode. Since the SIMD8 message applies to only 8 pixels, only the low 8 bits within this field are used. The high 8 bits are always set to 1. **[DevBW, DevCL]** Errata: Active Pixel Mask needs to be ORed with the inverse of the EMask before it is ANDed with the DMask. Also if the sample instruction is within a conditional then the active pixel mask will be overwritten with the partial mask on each different sample instruction so this will have to be done for each instance of the sample instruction not just as the end.

2.12.2.3 SIMD4x2

A SIMD4x2 writeback message always consists of a single message register containing all four channels of each of the two "pixels" (called "samples" here, as they are not really pixels) of data. The write channel mask bits as well as the execution mask on the "send" instruction are used to determine which of the channels in the destination register are overwritten. If any of the four execution mask bits for a sample is asserted, that sample is considered to be active. The active channels in the write channel mask will be written in the destination register for that sample. If the sample is inactive (all four execution mask bits deasserted), none of the channels for that sample will be written in the destination register.

DWord	Bit	Description
W0.7	31:0	**Sample 1 Alpha:** Specifies the value of the pixel's alpha channel. Format = IEEE Float, S31 signed 2's comp integer, or U32 unsigned integer. Format depends on the **Data Return Format** programmed for the surface being sampled.
W0.6	31:0	**Sample 1 Blue**
W0.5	31:0	**Sample 1 Green**
W0.4	31:0	**Sample 1 Red**
W0.3	31:0	**Sample 0 Alpha**
W0.2	31:0	**Sample 0 Blue**
W0.1	31:0	**Sample 0 Green**
W0.0	31:0	**Sample 0 Red**

2.12.2.4 SIMD32/64

2.12.2.4.1 sample_unorm* * [DevSNB]

Pixels are numbered as follows:

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

[DevSNB+]: Which registers are returned is determined by the write channel mask received in the corresponding input message. Each asserted write channel mask results in both destination registers of the corresponding channel being skipped in the writeback message, and all channels with higher numbered registers being dropped down to fill in the space occupied by the masked channel. For example, if only red and alpha are enabled, red is sent to regid+0 and regid+1, and alpha to regid+2 and regid+3 (using 16 bit Full mode as an example).

DWord	Bit	Description
W0.7	31:16	**Pixel 15 Red** Format = 16-bit UNORM with an 8-bit range (the value FF00h maps to a real value of 1.0) Range = [0000h:FF00h]
	15:0	**Pixel 14 Red**
W0.6		**Pixel 13 & 12 Red**

DWord	Bit	Description
W0.5		**Pixel 7 & 6 Red**
W0.4		**Pixel 5 & 4 Red**
W0.3		**Pixel 11 & 10 Red**
W0.2		**Pixel 9 & 8 Red**
W0.1		**Pixel 3 & 2 Red**
W0.0		**Pixel 1 & 0 Red**
W1.7		**Pixel 31 & 30 Red**
W1.6		**Pixel 29 & 28 Red**
W1.5		**Pixel 23 & 22 Red**
W1.4		**Pixel 21 & 20 Red**
W1.3		**Pixel 27 & 26 Red**
W1.2		**Pixel 25 & 24 Red**
W1.1		**Pixel 19 & 18 Red**
W1.0		**Pixel 17 & 16 Red**
W2.7:0		**Pixels 15:0 Green**
W3.7:0		**Pixels 31:16 Green**
W4.7:0		**Pixels 15:0 Blue** W4-W7 are not sent for the _RG versions of the sample_unorm message
W5.7:0		**Pixels 31:16 Blue** W4-W7 are not sent for the _RG versions of the sample_unorm message
W6.7:0		**Pixels 15:0 Alpha** W2 and W3 are not sent for the _RG versions of the sample_unorm message
W7.7:0		**Pixels 31:16 Alpha** W4-W7 are not sent for the _RG versions of the sample_unorm message

For the sample_unorm_RG+killpix and sample_unorm+killpix messages, an additional writeback phase is returned. For sample_unorm_RG+killpix, “n” is equal to 4, for sample_unorm+killpix, “n” depends on which channels are enabled for return, this register will immediately follow the first part of the writeback message.

DWord	Bit	Description
Wn.7:1		Reserved (not written)
Wn.0	31:0	**Active Pixel Mask:** This field has the bit for all pixels set to 1 except those pixels that have been killed as a result of chroma key with kill pixel mode. The bits in this mask correspond to the pixels as follows: 0 1 4 5 16 17 20 21 2 3 6 7 18 19 22 23 8 9 12 13 24 25 28 29 10 11 14 15 26 27 30 31

2.12.2.5 Sample_8x8 Writeback Messages

2.12.2.5.1 Sample_8x8 Writeback Messages [DevSNB]

The writeback message for sample_8x8 consists of up to 16 destination registers. Which registers are returned is determined by the write channel mask received in the corresponding input message. Each asserted write channel mask results in all four destination registers of the corresponding channel being skipped in the writeback message, and all channels with higher numbered registers being dropped down to fill in the space occupied by the masked channel.

Pixels are numbered as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63

"16 bit Full" Output Format Control Mode

DWord	Bit	Description
W0.7	31:16	**Pixel 15 Red** Format = 16-bit UNORM with an 8-bit range (the value FF00h maps to a real value of 1.0) Range = [0000h:FF00h]
	15:0	**Pixel 14 Red**
W0.6		**Pixel 13 & 12 Red**
W0.5		**Pixel 7 & 6 Red**
W0.4		**Pixel 5 & 4 Red**
W0.3		**Pixel 11 & 10 Red**
W0.2		**Pixel 9 & 8 Red**
W0.1		**Pixel 3 & 2 Red**
W0.0		**Pixel 1 & 0 Red**
W1.7		**Pixel 31 & 30 Red**
W1.6		**Pixel 29 & 28 Red**
W1.5		**Pixel 23 & 22 Red**
W1.4		**Pixel 21 & 20 Red**
W1.3		**Pixel 27 & 26 Red**
W1.2		**Pixel 25 & 24 Red**
W1.1		**Pixel 19 & 18 Red**
W1.0		**Pixel 17 & 16 Red**
W2.7:0		**Pixels 15:0 Green**
W3.7:0		**Pixels 31:16 Green**
W4.7:0		**Pixels 15:0 Blue**
W5.7:0		**Pixels 31:16 Blue**

DWord	Bit	Description
W6.7:0		**Pixels 15:0 Alpha**
W7.7:0		**Pixels 31:16 Alpha**
W8.7:0		**Pixels 47:32 Red**
W9.7:0		**Pixels 63:33 Red**
W10.7:0		**Pixels 47:32 Green**
W11.7:0		**Pixels 63:33 Green**
W12.7:0		**Pixels 47:32 Blue**
W13.7:0		**Pixels 63:33 Blue**
W14.7:0		**Pixels 47:32 Alpha**
W15.7:0		**Pixels 63:33 Alpha**

2.12.2.5.2 deinterlace

The deinterlace message has three different writeback messages, depending on the **DI Enable** and **DI Partial** fields of SAMPLER_STATE.

Pixels are indicated by an (X, Y) pair. The following tables indicate the format of common **Luma, Chroma, STMM,** and **Block Noise Estimate/Denoise History** blocks defined as portions of the specific writeback messages defined in the following sections. Each block defines one register.

Luma block definition:

DWord	Bit	Description
Wn.7	31:24	**Luma (15,1)** Format = U8
	23:16	**Luma (14,1)**
	15:8	**Luma (13,1)**
	7:0	**Luma (12,1)**
Wn.6	31:0	**Luma (11:8,1)**
Wn.5	31:0	**Luma (7:4,1)**
Wn.4	31:0	**Luma (3:0,1)**

DWord	Bit	Description
Wn.3	31:0	**Luma (15:12,0)**
Wn.2	31:0	**Luma (11:8,0)**
Wn.1	31:0	**Luma (7:4,0)**
Wn.0	31:0	**Luma (3:0,0)**

Chroma block definition:

DWord	Bit	Description
Wp.7	31:24	**Cb (14,1)** Format = U8
	23:16	**Cr (14,1)** Format = U8
	15:8	**Cb (12,1)**
	7:0	**Cr (12,1)**
Wp.6	31:0	**Cr & Cb (10:8,1)**
Wp.5	31:0	**Cr & Cb (6:4,1)**
Wp.4	31:0	**Cr & Cb (2:0,1)**
Wp.3	31:0	**Cr & Cb (14:12,0)**
Wp.2	31:0	**Cr & Cb (10:8,0)**
Wp.1	31:0	**Cr & Cb (6:4,0)**
Wp.0	31:0	**Cr & Cb (2:0,0)**

STMM block definition:

DWord	Bit	Description
Wr.7	31:24	**STMM (14,3)** Format = U8
	23:16	**STMM (12,3)**
	15:8	**STMM (10,3)**

DWord	Bit	Description
	7:0	**STMM (8,3)**
Wr.6	31:0	**STMM (6:0,3)**
Wr.5	31:0	**STMM (14:8,2)**
Wr.4	31:0	**STMM (6:0,2)**
Wr.3	31:0	**STMM (14:8,1)**
Wr.2	31:0	**STMM (6:0,1)**
Wr.1	31:0	**STMM (14:8,0)**
Wr.0	31:0	**STMM (6:0,0)**

Block Noise Estimate/Denoise History block definition: [DevSNB DI enabled]

DWord	Bit	Description
Wq.7	31:16	Y[15:0] – Location of 16x4
Wq.7	15:0	X[15:0] - Location of 16x4
Wq.6	31:24	STAD0 - Sum in time of absolute differences for 4x4 Format = U8 [STAD values are 0 if DN is disabled]
Wq.6	23:16	STAD1
Wq.6	15:8	STAD2
Wq.6	7:0	STAD3 (Ignore when both DN & DI are enabled)
Wq.5	31:24	SHCM0 - Sum horizontally of absolute differences for 4x4 Format = U8 [SHCM values are 0 if DN is disabled]
Wq.5	23:16	SHCM1
Wq.5	15:8	SHCM2
Wq.5	7:0	SHCM3 (Ignore when both DN & DI are enabled)
Wq.4	31:24	SVCM0 Sum Vertically of absolute differences for 4x4 Format = U8 [SVCM values are 0 if DN is disabled]
Wq.4	23:16	SVCM1

DWord	Bit	Description
Wq.4	15:8	SVCM2
Wq.4	7:0	SVCM3 (Ignore when both DN & DI are enabled)
Wq.3	31:16	Diff_cTpT - difference in top fields of current and previous frame Format = U16
Wq.3	15:0	Diff_cBpB - difference in bottom field of current and previous frame
Wq.2	31:16	Diff_cTcB - difference between top and bottom field in current frame.
Wq.2	15:0	Diff_cTpB - difference between current top and previous bottom
Wq.1	31:16	Diff_cBpT - difference between current bottom and previous top.
Wq.1	15:8	Motion_Count - number of pixels that are moving (different above a threshold) Format = U8
Wq.1	7:0	Block Noise Estimate for 16x4 (Valid only if DN is enabled)
Wq.0	31:24	**Denoise History** for 4x4 at Y = 15 to 12, X = 3 to 0 Format = U8
Wq.0	23:16	**Denoise History** for 4x4 at Y = 11 to 8, X = 3 to 0
Wq.0	15:8	**Denoise History** for 4x4 at Y = 7 to 4, X = 3 to 0
Wq.0	7:0	**Denoise History** for 4x4 at Y = 3 to 0, X = 3 to 0

Block Noise Estimate/Denoise History block definition: [DevSNB DI disabled]

DWord	Bit	Description
Wq.7	31:16	Y[15:0] – Location of 16x4
Wq.7	15:0	X[15:0] - Location of 16x4
Wq.6	31:24	STAD0 - Sum in time of absolute differences for 4x8 Format = U8
Wq.6	23:16	STAD1
Wq.6	15:8	STAD2
Wq.6	7:0	STAD3

DWord	Bit	Description
Wq.5	31:24	SHCM0 - Sum horizontally of absolute difference for 4x8
Wq.5	23:16	SHCM1
Wq.5	15:8	SHCM2
Wq.5	7:0	SHCM3
Wq.4	31:24	SVCM0 Sum Vertically of absolute difference for 4x8
Wq.4	23:16	SVCM1
Wq.4	15:8	SVCM2
Wq.4	7:0	SVCM3
Wq.3	31:16	Reserved
Wq.3	15:0	Reserved
Wq.2	31:8	Reserved
Wq.2	7:0	Block Noise Estimate for 16x8
Wq.1	31:24	**Denoise History** for 4x4 at X = 15 to 12, Y = 7 to 4 Format = U8
Wq.1	23:16	**Denoise History** for 4x4 at X = 11 to 8, Y = 7 to 4
Wq.1	15:8	**Denoise History** for 4x4 at X = 7 to 4, Y = 7 to 4
Wq.1	7:0	**Denoise History** for 4x4 at X = 15 to 12, Y = 3 to 0
Wq.0	31:24	**Denoise History** for 4x4 at Y = 15 to 12, X = 3 to 0 Format = U8
Wq.0	23:16	**Denoise History** for 4x4 at Y = 11 to 8, X = 3 to 0
Wq.0	15:8	**Denoise History** for 4x4 at Y = 7 to 4, X = 3 to 0
Wq.0	7:0	**Denoise History** for 4x4 at Y = 3 to 0, X = 3 to 0

DI Enabled (Only)

This writeback message is returned when the DI Enable field in SAMPLER_STATE is enabled. The response length possibilities are:

- DevSNB & DN Enabled: 12

- DevSNB & DN Disabled: 10

DWord	Bit	Description
W0		**Previous 2nd Field Deinterlaced Luma for Y=0,1** Refer to Luma block above for definition.
W1		**Previous 2nd Field Deinterlaced Luma for Y=2,3**
W2		**Previous 2nd Field Deinterlaced Chroma for Y=0,1** Refer to Chroma block above for definition.
W3		**Previous 2nd Field Deinterlaced Chroma for Y=2,3**
W4		**Current 1st Field Deinterlaced Luma for Y=0,1**
W5		**Current 1st Field Deinterlaced Luma for Y=2,3**
W6		**Current 1st Field Deinterlaced Chroma for Y=0,1**
W7		**Current 1st Field Deinterlaced Chroma for Y=2,3**
W8		**STMM** Refer to STMM block above for definition.
W9		**Block Noise Estimate/Denoise History** Refer to Block Noise Estimate/Denoise History block above for definition.
W10		**Current 2nd Field Luma for 16x2** This register is only included if **DN Enable** is enabled.
W11		**Current 2nd Field Chroma** This register is only included if **DN Enable** is enabled. Only valid if input surface format is 4:2:2

The denoised luma for both the current 1st and 2nd field needs to be written out, but only the 2nd field has a dedicated location. This is because the denoised data for the 1st field is in the deinterlaced output for the 1st field – Y=0 and Y=2 are the denoised data, while Y=1 and Y=3 either the deinterlaced lines or copied from the previous or current frame if progressive.

DI Disabled

This writeback message is returned when the **DI Enable** field in SAMPLER_STATE is disabled. The DN with DI disabled responses with a 16x8 block rather than a 16x4 with a response length of 9 for a 4:2:2 input format, or 5 for other formats. Two denoised luma and chroma fields are combined into an interleaved top/bottom format.

		Description
W0		**Luma for Y=0 & 1** Refer to Luma block above for definition.
W1		**Luma for Y=2 & 3** Refer to Luma block above for definition, but add 2 to Y to get location

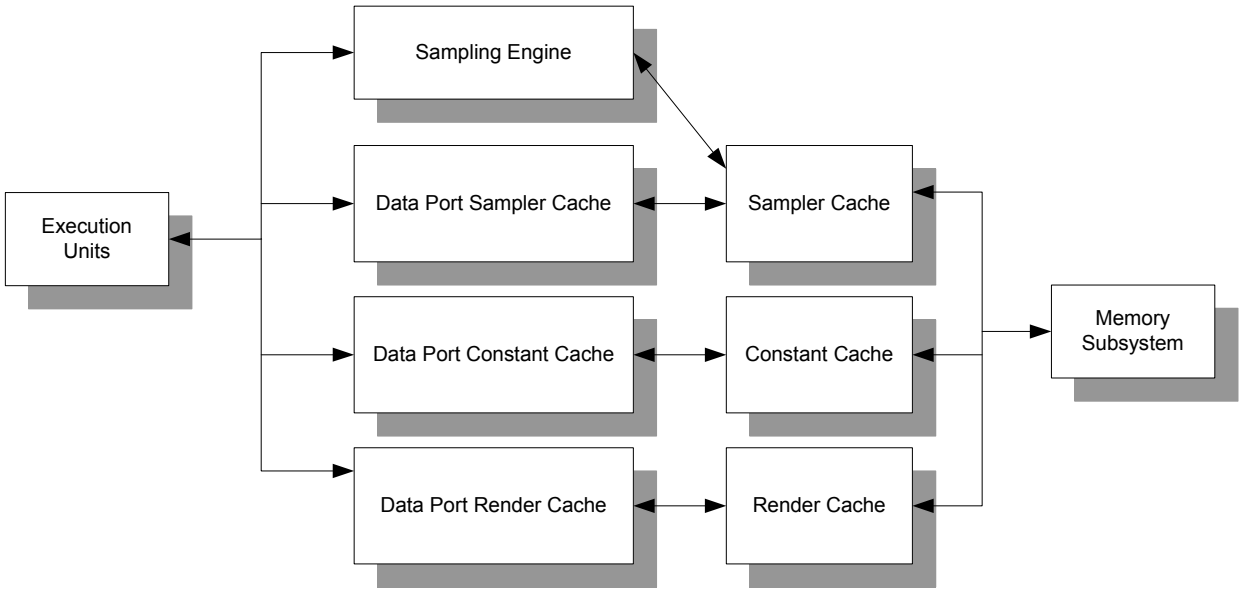
		Description
W2		**Luma for Y=4 & 5**
W3		**Luma for Y=6 & 7**
W4.7	31:16	**Y[15:0]** Y co-ordinate of the current block within the frame
W4.7	15:0	**X[15:0]** X co-ordinate of the current block within the frame
W4.6	31:24	**STAD0** – Sum in time of absolute differences for the 1st 4x8 Format = U8
W4.6	23:16	**STAD1**– Sum in time of absolute differences for the 2nd 4x8
W4.6	15:8	**STAD2 –** Sum in time of absolute differences for the 3rd 4x8
W4.6	7:0	**STAD3** – Sum in time of aboslute differences for the 4th 4x8
W4.5	31:24	**SHCM0** – Sum horizontaly of absolute differences
W4.5	23:16	**SHCM1**
W4.5	15:8	**SHCM2**
W4.5	7:0	**SHCM3**
W4.4	31:24	**SVCM0** – Sum vertically of absolute differences.
W4.4	23:16	**SVCH1**
W4.4	15:8	**SVCH2**
W4.4	7:0	**SVCH3**
W4.3	31:0	Reserved : MBZ
W4.2	31:8	Reserved : MBZ
	7:0	**Block Noise Estimate** Format = U8
W4.1	31:24	**Denoise History for 4x4 at X = 15 to 12, Y = 7 to 4**
	23:16	**Denoise History for 4x4 at X = 11 to 8, Y = 7 to 4**
	15:8	**Denoise History for 4x4 at X = 7 to 4, Y = 7 to 4**
	7:0	**Denoise History for 4x4 at X = 3 to 0, Y = 7 to 4**

		Description
W4.0	31:24	**Denoise History for 4x4 at X = 15 to 12, Y = 3 to 0**
	23:16	**Denoise History for 4x4 at X = 11 to 8, Y = 3 to 0**
	15:8	**Denoise History for 4x4 at X = 7 to 4, Y = 3 to 0**
	7:0	**Denoise History for 4x4 at X = 3 to 0, Y = 3 to 0**
W5		**Chroma for Y=0 & 1** Refer to Chroma block above for definition. Only delivered if input surface format is 4:2:2
W6		**Chroma for Y=2 & 3** Refer to Chroma block above for definition, but add 2 to Y to get location. Only delivered if input surface format is 4:2:2
W7		**Chroma for Y=4 & 5** Only valid if input surface format is 4:2:2
W8		**Chroma for Y=6 & 7** Only sent if input surface format is 4:2:2

3. Data Port

The Data Port provides all memory accesses for the DevSNB subsystem other than those provided by the sampling engine. These include render target writes, constant buffer reads, scratch space reads/writes, and media surface accesses.

[DevSNB+]: The diagram below shows the three parts of the Data Port (Sampler Cache, Constant Cache, and Render Cache) and how they connect with the caches and memory subsystem. The execution units and sampling engine are shown for clarity.



The kernel programs running in the execution units communicate with the data port via messages, the same as for the other shared function units. The three data ports are considered to be separate shared functions, each with its own shared function identifier.

3.1 Cache Agents

The data port allows access to memory via various caches. The choice of which cache to use for a given application is dictated by its restrictions, coherency issues, and how heavily that cache is used for other purposes.

[DevSNB+]: The cache to use is selected by the shared function accessed.

3.1.1 Render Cache

[DevSNB]: The render cache is the only cache that supports both reads and writes. All writes must use this cache. In addition, all reads to a surface that is also being written should use this cache to avoid expensive flushing that would be required for coherency. The render cache supports both linear and tiled memory.

The render cache is intended to be used for the following surfaces:

- 3D render target surfaces
- destination surfaces for media applications
- intermediate working surfaces for media applications
- scratch space buffers
- streamed vertex buffers

3.1.2 Sampler Cache

The sampler cache is a read-only cache that supports both linear and tiled memory. In addition to being used by the sampling engine (via the sampling engine messages), the sampler cache is intended to be used for source surfaces in media applications via the data port. The same application may use the sampler cache via the sampling engine and data port without flushing the pipeline between accesses.

3.1.3 Constant Cache [DevSNB+]

The constant cache is a read-only cache that supports only linear memory and only the messages that operate on buffer surface types. It is intended to be used only for constant buffers.

3.2 Surfaces

The data elements accessed by the data port are called “surfaces”. There are two models used by the data port to access these surfaces: surface state model and stateless model.

3.2.1 Surface State Model

The data port uses the binding table to bind indices to surface state, using the same mechanism used by the sampling engine. The surface state model is used when a **Binding Table Index** (specified in the message descriptor) of less than 255 is specified. In this model, the **Binding Table Index** is used to index into the binding table, and the binding table entry contains a pointer to the SURFACE_STATE. SURFACE_STATE contains the parameters defining the surface to be accessed, including its location, format, and size.

This model is intended to be used for constant buffers, render target surfaces, and media surfaces.

3.2.2 Stateless Model

The stateless model is used when a **Binding Table Index** (specified in the message descriptor) of 255 is specified. In this model, the binding table is not accessed, and the parameters that define the surface state are overloaded as follows:

- Surface Type = SURFTYPE_BUFFER
- Surface Format = R32G32B32A32_FLOAT
- Vertical Line Stride = 0
- Surface Base Address = **General State Base Address** + **Immediate Base Address**
- Buffer Size = checked only against **General State Access Upper Bound**
- Surface Pitch = 16 bytes
- Utilize Fence = false
- Tiled = false

This model is primarily intended to be used for scratch space buffers.

3.3 Write Commit

For write messages, an optional write commit writeback message can be requested via the Send Write Commit Message bit in the message descriptor. This bit causes a return message to the thread indicating when the write has been committed to the in-order cache pipeline and it is safe to issue another access to the same data with the assurance that it will happen after the first write. A read issued after the write commit ensures that the read will get the newly written data, and another write issued after the write commit will be the last to modify the data. "Committed" does not guarantee that the data has been actually written to the memory subsystem, but only that the write has been scheduled and cannot be passed by another read or write issued subsequently.

If **Send Write Commit Message** is used on a Flush Render Cache message, the write commit is sent only when the render cache has completed its flush to memory. A read issued to another cache after the write commit is received will be guaranteed to retrieve the “new” data that was written before the Flush Render Cache message was issued.

The write commit does not modify the destination register, but merely clears the dependency associated with the destination register. Thus, a simple “mov” instruction using the register as a source is sufficient to wait for the write commit to occur. The following code sequence indicates this:

```
send r12 m1 DPWRITE     ; issue write to render cache

mov m1 r3               ; assemble read message

mov r12 r12             ; block on write commit

send r13 m1 DPREAD      ; read same location as write
```

[DevSNB-A] Erratum: A write message with all the addresses/offsets out of bounds with write-commit bit set is not supported.

[DevSNB] Prior to End of Thread with a URB_WRITE, the kernel must ensure all writes are complete by sending the final write as a committed write for all non-pixel shaders.

3.4 Read/Write Ordering

[DevSNB+]: Reads and writes issued from the same thread *are* guaranteed to be processed in the same order as they are issued. Software mechanisms must still ensure ordering of accesses issued from different threads.

3.5 Accessing Buffers

There are four data port messages used to access buffers. Three of these are used for both constant buffers and scratch space buffers, the fourth is used by the geometry shader kernel to write to streamed vertex buffers. All of these messages support only buffers, and can use the surface state model as well as the stateless model.

The following table indicates the intended applications of each of the buffer messages.

Message	Applications
OWord Block Read/Write	• constant buffer reads of a single constant or multiple contiguous constants • scratch space reads/writes where the index for each pixel/vertex is the same • block constant reads, scratch memory reads/writes for media
OWord Dual Block Read/Write	• SIMD4x2 constant buffer reads where the indices of each vertex/pixel are different (if there are two indices and they are the same, hardware will optimize the cache accesses and do only one cache access) • SIMD4x2 scratch space reads/writes where the indices are different.
DWord Scattered Read/Write	• SIMD8/16 constant buffer reads where the indices of each pixel are different (read one channel per message) • SIMD8/16 scratch space reads/writes where the indices are different (read/write one channel per message) • general purpose DWord scatter/gathering, used by media
Streamed Vertex Buffer Write	• geometry shader streaming vertex data out

These messages generally ignore the surface format field of the state and perform no format conversion. The exception is the Streamed Vertex Buffer Write, which uses the surface format field to determine only how many channels are to be written. The data contained in each channel is still not converted in any way.

3.6 Accessing Media Surfaces

The Media Block Read/Write message is intended to be used to access 2D media surfaces. The message specifies an X/Y coordinate into the 2D surface as input. Since this message only supports 2D surfaces, the stateless model cannot be used with this message.

3.6.1 Color Processing [DevSNB+]

The image enhancement color processing pipe, known as IECP or shortly CP. The pipe contains a couple of functions:

- Packer with 422 to 444 converter.
- Skin Tone detection & Enhancement (STDE).
- TCCE - Automatic Contrast Enhancement (ACE) & Total Color Control (TCC).
- Procamp.
- Color Space Converter (CSC).
- repacker with 444 to 422 converter

Since these functions are performed on per-pixel basis, IECP is integrated in Render Cache Pixel Backend (RCPB). The operation of each functionality could be on/off through the enable bit of each function.

Note: all of the state parameters related to IECP are denoted in the bold and italic font format.

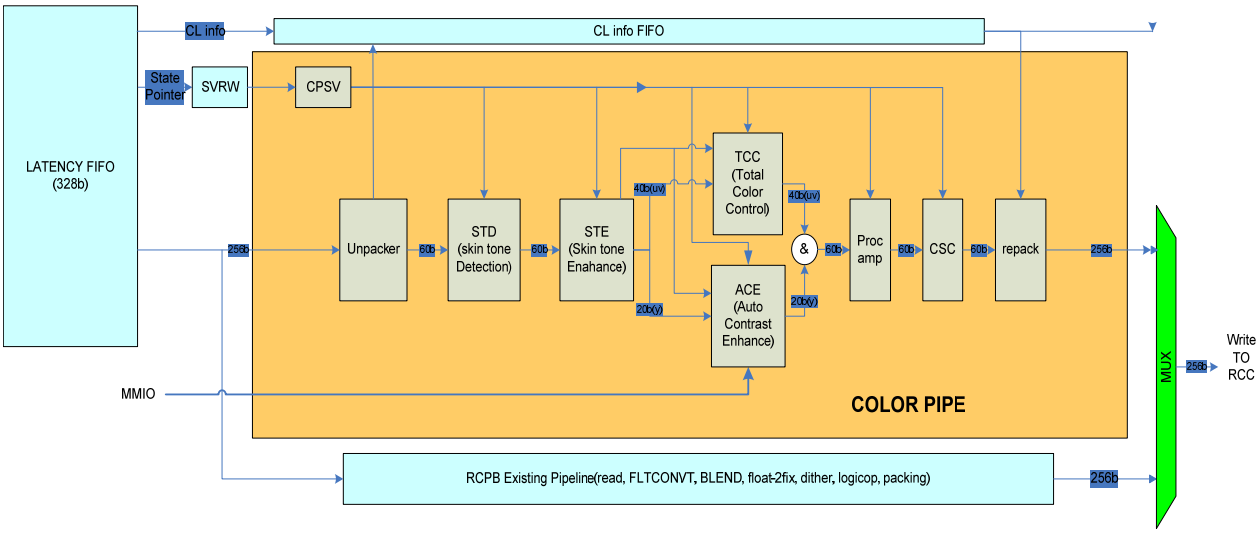
3.6.1.1 Overview of color processing pipeline

The input message to IECP is 256 bits data from RCPB (contains 2 lines X 2 pixels per clock). This **unpacker** converts 256b into two pixels per clock, 36 bits each. In case of 422 inputs the UV are the same for the two pixels in the pair (422 to 444 conversion).

The **Re-packer** (the CSC) delivers 2 pixels in parallel, 36 bits each. The 2x2 message pixels are packed again to 256b and sent with the outgoing message. The 256 bits are organized according to the data type (422/444, 8/16 bits). In case of 422 output, the UV is the average of two adjacent pixels. Also the pipe itself is 12bis/pixel component, in the output message it will be either 8 bit/pixel component (while taking only the 8 MSB) or 16 bits/pixel component (while adding 0000 at the LSB).

There is statistic information from ACE block (10 bit histogram, 1 bit aoi and 1bit skin pixel) to be sent to VSC (Video Statistic Counter). VSC will process on these data and output the maximum and minimum value of the luma values (Ymax and Ymin) and the number of total skin pixels through MMIO. The

Software development can access these data through MMIO and performs the SW part of the color processing algorithms.



The color-processing enables the user to customize visual quality of video playback on the PC platform. The seven functions main goals can be summarized as:

(i) 422 to 444 converter and the 444 to 422 converter functions enable us some flexibility in the data format input and output.

(ii) Skin Tone Detection/Enhancement function detects skin like color and attempts to change the tone based on user specified parameters to make it more palatable to the user.

(iii) Automatic Contrast Enhancement increases details in dark and bright areas by changing the contrast function in relation to frames luma histogram.

(iv) Total color control allows the user to increase or decrease the color saturation of the six basic colors (Red, Green, Blue, Magenta, Cyan, Yellow).

(v) Procamp enables the user to control the Brightness, Contrast, Saturation and the Hue.

(vi) Color Space Converter enables the user to convert color space from YUV format to RGB.

3.6.1.2 Skin Tone Detection/Enhancement (STD/E)

The STD/E unit, composed of the Skin Tone Detection (STD) and Skin Tone Enhancement (STE) units, is part of color processing pipe located at the Render Cache Pixel Backend (RCBP).

The main goal of the STD/E is to reproduce the skin colors in a way that is more palatable to the observer, and by that to increase the sensed image quality. It may also pass indication of skin tones to the TCC and ACE.

The STD unit detects the skin like colors and passes a grade of skin tone color to the STE. The STE modify the saturation and Hue of the pixel. Both the STD and STE are per-pixel basis. The input pixels are required to be on the YUV space.

The skin tone detected factor will be recorded as a 5-bit number and it will be passed to the module of ACE and TCC to indicate the strength of skin tone likelihood.

3.6.1.2.1 STD

The STD operates on digital images in the YUV color space. In these space the skin-tone region is represented by the ellipse in the (U,V) subspace (chroma components), by a trapeze membership function in the Y direction (luma component) and by a piece-wise linear classifier in the (V,Y) subspace.

U,V data is transformed into Hue and Saturation space through shifting and rotation

Step 1: shift rectangle

```
U_center =  U – U_mid
V_center =  V - V_mid
```

Step 2: rotate rectangle

```
Sat = -(U_center * Cos - V_center * Sin)
Hue = -(U_center * Sin + V_center * Cos)
```

Where: Sin = ***Sin(α),*** and Cos = ***Cos(α).***

Rectangle skin-tone measure determination

Skin-tone detection is described by a continue score on the [0,1] range, where a level 0 means not a skin (SkinToneFactor = 0) , and a level 1 (SkinToneFactor = 1) means a full skin. In between, (0,1) region, we have partial skin-tone detection. This partial skin-tone detection is controlled by a margin parameter, which will be denoted by "***HS_margin***". The SkinToneFactor is expressed by 5 bits, and thus have values in the [0,31] range.

```
if( abs(Sat) <= SatMax && abs(Hue) <= HueMax)
  {
   if(HS_margin >= 5)
      {
        Sat_Factor = (Sat_max-abs(Sat)) / 2^(HS_margin - 5);
```

```
        Hue_Factor = (Hue_max-abs(Hue)) / 2^(HS_margin - 5);
      }
   else
      {
        Sat_Factor = (Sat_max-abs(Sat)) * 2^(HS_margin - 5);
        Hue_Factor = (Hue_max-abs(Hue)) * 2^(HS_margin - 5);
      } //end of if(HS_margin >= 5)
 }
else
{
        Sat_Factor = 0;
        Hue_Factor = 0;
} //end of if( abs(Sat) <= SatMax && abs(Hue) <= HueMax)

Sat_Factor = min(Sat_Factor,31);
Hue_Factor = min(Hue_Factor,31);

Rectagle_SkinToneFactor = min(Sat_Factor, Hue_Factor);
```

Rhombus skin tone detection determination

Similar to the rectangle skin-tone measure, a rhombus-margin (***Diamond_margin***) is introduced. This introduces a new rhombus region, inner to the original rhombus, in a similar happened with the rectangle. There are two regions such that: outside the original rhombus a SkinToneFactor = 0 (not a skin); inside the inner rhombus SkinToneFactor = 1 (full skin); in between 0 < SkinToneFactor < 1 indicating a partial skin-tone detection. As in the rectangle case, the SkinToneFactor is expressed by 5 bits, and thus have values in the [0,31] range.

A Diamond SkinToneFactor calculations algorithm is:

```
Dist = abs(Sat – Diamond_du) + Diamond_alpha(1/tan(β)) * abs(Hue –
Diamond_dv);


                    //outside the diamond

if(Dist >= Diamond_TH)

{

  D_Factor = 0; //the point is out of the large rhombus

}

else if(Dist < (Diamond_TH - Diamond_margin))

  {

    D_Factor = 31; //the point is inside the inner rhombus

  }

   else  //the point is inbetween the outer and the inner rhombuses

   {

    if(Diamond_margin >= 5)

     {

      D_Factor = (Diamond_TH - Dist) / 2^(Diamond_margin - 5);

     }

     else

     {

      D_Factor = (Diamond_TH - Dist) * 2^(Diamond_margin - 5);

     } // end of if(Diamond_margin >= 5)

   } // if(D < (Diamond_TH - Diamond_margin))
```

```
Diamond_SkinToneFactor  = D_factor;
```

Finally the level of the skin-tone detection in the (U,V) subspace is given by:

UV_SkinToneFactor = min(Rectangle_SkinToneFactor, Diamond_SkinToneFactor);

Detection in Y direction

The detection based on the Y-values, is given by a piece-wise linear membership function, which is defined with 4 points ***(Y_point_x***) (x=1, 2, 3, and 4).

```
if(Y >= Y_Point_0 && in_Y < Point_1)
   Y_Factor = (Y – Y_Point_0) * Y_Slope_1;
else if(Y >= Point_1 && Y < Point_2)
        Y_Factor = 31;
     else if(Y >= Point_2  && Y < Point_3)
             Y_Factor = (Point_3 - Y) * Y_Slope_2 ;
          else
             Y_Factor = 0;
```

At the end of the process a double (min,max) clipping is applied:

```
Y_Factor = min(31,max(Y_Factor,0));
```

The final Skin-Tone detection is is given by:

```
SkinToneFactor = min(UV_SkinToneFactor, Y_factor);
```

Detection in the VY plane (3D-like DTD)

The operation of the detection in VY plane is particularly enabled by ***VY_STD_Enable*** bit

It is known that the application of a three-dimensional (3D) classifier in the (Y,U,V) space, instead of a two dimensional (2D) skin-tone detector in the (U,V) plane, is resulted in a better detection. Implementation complexity of the full 3D classifier is too high, and forces us to approximate the classifier by more simple, but useful methods. Skin-tone data distribution implies (it is almost convex, and has a predominate directions) that the 3D classifier could be approximated by the intersection of the three 2D classifiers in (U,V), (U,Y), and (V,Y) subspaces. The (U,V) subspace is the most important one it is already approximated by the ellipse, as was described previously. Our study implies that the (V,Y) subspace is the next most important one. Although the (U,Y) space carries the STD information, it is heavily redundant and has the reduced importance.

Thus the approximation of 3D classifier is an intersection of (U,V) and (V,Y) two-dimensional classifiers. The (V,Y) classifier is given by two piece-wise linear functions (PWLF), Each PWLF is composed of four straight segments. Each segment is described by the three parameters (Point, Slope and bias). Thus a single PWLF (lower or upper) is described by 12 parameters (4 points, 4 biases, 4 slopes).

The parameters of lower part are: 4 point ***PxL*** (x=0, 1, 2, 3), 4 bias ***BxL*** (x=0, 1, 2, 3) and 4 slope ***SxL*** (x=0, 1, 2, 3).

The parameters of upper part are: 4 point ***PxU*** (x=0, 1, 2, 3), 4 bias ***BxU*** (x=0, 1, 2, 3) and 4 slope ***SxU*** (x=0, 1, 2, 3).

There are Programming Restrictions to specify the parameters:

- The points must be in the non-decreasing order: P0 <= P1 <= P2 <= P3.
- The parts must be continues on they ends. Thus the user:

  (a). must set: $P0_L = P0_U$ (continuity at the leftmost points).

  (b). must care for continuity at the rightmost points.

Margin for the detection in the VY plane (3D-like DTD)

Vertical margins of each part were introduced to provide a “soft” continuous detection over the classifier boundaries. There are two parameters defined

MarginVYL - the margin of the lower (blue) part.

MarginVYU - the margin of the upper (red) part.

Consider a pixel with coordinates (Y,V) = $(P2_L,V_1)$,. This pixel has a Y coordinate exactly as of the point P2, and a V coordinate equal V1. For this pixel the detection relative to the Lower Part will be:

detL = Min (Max ((V1 – B2L) / MarginVYL, 0), 1)

The identical calculations are made for the Upper Line as well:

detU = Min (Max ((VU – V1) / MarginVYU, 0) , 1)

Where:

det_L - is a detection relative to the Lower Part

det_U - is a detection relative to the Upper Part

V_U - is a V value of the Upper PWLF correspond to the $Y=P2_L$

B_U - is a V value of the Lower PWLF correspond to the $Y=P2_L$

The inverse operation of (1/ MarginVYL), and (1/ MarginVYU) is specified by the parameters INV***_MARGIN_VYL*** and ***INV_Margin_VYU***.

Both detections (det_L, det_U) are reduced to 5 bit representations, and the overal detection in the (V,Y)-plane is given by:

$$det_VY = \min(det_L, det_U)$$

The final Skin-Tone Detection is given by the minimum of the previously calculated STD in the (U,V)-plane (9), and the current one:

SkinToneFactor = min(SkinToneFactor, det_VY)

This value is represented with 5 bits, and has a [0,31] range.

3.6.1.2.2 STE

The enhancement step is performed on the pixels which were detected as the skin-tone pixels only by the previous (STD) step. This step is divided into two sub-steps: saturation correction enhancemen and hue correction enhancement

STE – Saturation Correction Enhancement

The enhancement is performed by the transformation $Sat_{New} = F_{Sat}(Sat_{Old})$, which is realized by the piece-wise linear function (PWLF) with a 4 straight segments.

The parameters of this PWLF are:

Points:

SATP0 = -SatMax

SATPx (x=1,2,3) – defined by the user

SATP4 = SatMax

Biases:

SATB0 = -SatMax

SATBx (x=1,2,3) – defined by the user

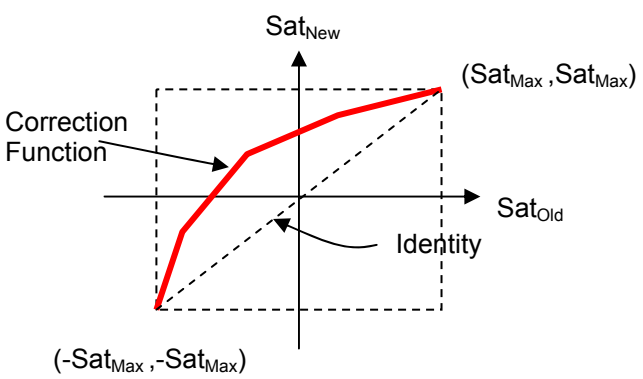
SATB4 = SatMax

Slopes:

SATSx (x=0,1,2,3) – defined by the user

There are Programming Restrictions to specify the parameters:

- The point Sat = $-Sat_{Max}$ maps to itself: $(-Sat_{Max}) \rightarrow (-Sat_{Max})$.
- The point Sat = Sat_{Max} maps to itself: $(Sat_{Max}) \rightarrow (Sat_{Max})$.
- The correction function is continuous.
- The correction function is non-decreasing.



General form of the Saturation correction PWLF

STE – Hue Correction Enhancement

The enhancement is performed by the transformation $Hue_{New} = F_{Sat}(Hue_{Old})$, which is realized by the piece-wise linear function (PWLF) with a 4 straight segments.

The parameters of this PWLF are:

Points:

HUEP0 = -HueMax

HUEPx (x=1,2,3) – defined by the user

HUEP4 = HueMax

Biases:

HUEB0 = -HueMax

HUEBx (x=1,2,3) – defined by the user

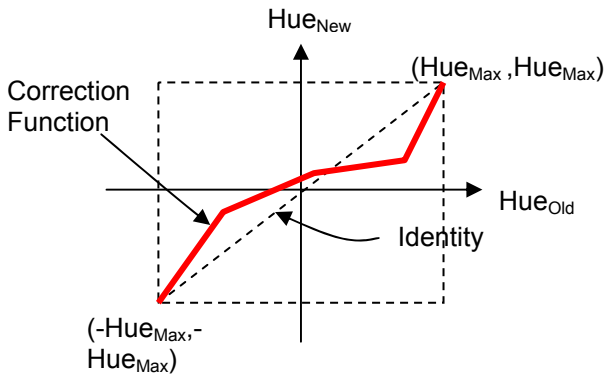
HUEB4 = HueMax

Slopes:

HUESx (x=0,1,2,3) – defined by the user

There is Programming Restrictions to specify the parameters

- The point Hue = $-HueUE_{Max}$ maps to itself: $(-Hue_{Max}) \rightarrow (-Hue_{Max})$.
- The point Hue = Hue_{Max} maps to itself: $(Hue_{Max}) \rightarrow (Hue_{Max})$.
- The correction function is continuous.
- The correction function is non-decreasing.



General form of the Hue correction PWLF

STE – Skin Type Correction Enhancement

The operation of this mode is enabled by the control parameter ***Skin_types_enable***.

The Saturation and Hue enhancement processes are basic STE procedure. The advanced mode to adjust the enhacement based on the skin type define the second set of the Sat and the Hue enhancement parameters, which has an identical structure as the previous one (Points, Biases, Slopes) but having different values. We will refer one set of parameters to the Bright skin (Bs), and the other to the Dark skin (Ds). Each pixel is referred as belongs to the Bright, the Dark, or to the both skin types with a different membership values. The Dark/Bright skin classifier is defined by the two parameters: ***Skin_types_thesh***, and ***Skin_types_margin***. It works on the luma (Y) values.

The parameters related are

Points:

HUEPx_DARK (x=1,2,3) – defined by the user

SATPx_DARK (x=1,2,3) – defined by the user

Biases:

HUEBx_DARK (x=1,2,3) – defined by the user

SATBx_DARK (x=1,2,3) – defined by the user

Slopes:

HUESx_DARK (x=0,1,2,3) – defined by the user

SATSx_DARK (x=0,1,2,3) – defined by the user

For the luma value Y, we define

Y_A = skinTypesThesh - skinTypesMargin

Y_B = skinTypesThesh + skinTypesMargin

$$MV_{Dark} = \begin{cases} 1, & \text{if } Y < Y_A \\ 0, & \text{if } Y > Y_B \\ (Y_B - Y) / (2^* \text{ skinTypesMargin}), & \text{if } Y_A <= Y <= Y_B \end{cases}$$

$$MV_{Bright} = 1 - mV_{Dark}$$

Where MV_{Dark} and MV_{Bright} are the membership value of the Dark and Bright skin (belongnes). *(Note: the membership values represent the "belongness" of the skin pixel to the different skin types).* Also, we mark that the inversee operation of 1/(2* Skin_types_margin) wil be specified by the parameter ***INV_skin_type_margin***.

In previous sections the procedure for the calculation of the Sat_{New} and Hue_{New} values was described. We calculate these values for the two skin types and thus get $Sat_{New\ B}$, $Hue_{New\ B}$, and $Sat_{New\ D}$, $Hue_{New\ D}$ values, where and subscribes "B" and "D" stands for the Bright and the Dark skin types, respectively. (In this case, the parameters with "_DARK" extension are used to work out $Sat_{New\ D}$ and $Hue_{New\ D}$, and the other set of the parameter could be reloaded with the parameters to work out $Sat_{New\ D}$, $Hue_{New\ D.}$)The final values of the enhanced pixel will be given by:

$$Sat_{New} = MV_{Dark} * Sat_{New\ D} + MV_{Bright} * Sat_{New\ B}$$

$$Hue_{New} = MV_{Dark} * Hue_{New\ D} + MV_{Bright} * Hue_{New\ B}$$

STE – (Sat, Hue) to (U, V) transformation

In prior session, the (U,V) → (Sat,Hue) transformation was proceeded by the two steps: ***shift,*** and ***rotation***. Thus the backward transformation should be done in the inverse order: a ***rotation***, and then a ***shift***.

```
// Rotate back:
U_Center_New =  (Sat_New * Cos) + (Hue_New * Sin)
V_Center_New = -(Sat_New * Sin) + (Hue_New * Cos)
```

```
// Shift:
            U_New =  U_Center_New + U_mid
            V_New =  V_Center_New + V_mid
```

The (U_new, V_new) are the (Sat_{New}, Hue_{New}) values in transformed to the original (U,V) coordinates.

Let denote the original (U,V) values of the pixel by (U_in,V_in). Thus the difference between the corrected and the original values are:

DU = U_new – U_in

DV = V_new – V_in

The final correction must be depended by the *SkinToneFactor* value, and therefore DU, DV are corrected by:

DU = DU * STD_ Likelihood_Factor

DV = DV * STD_ Likelihood_Factor

Where:

STD_ Likelihood_Factor = (SkinToneFactor / 32)

(Remember that the 0 <= SkinToneFactor <= 31).

After the DU and DV were corrected by the STD likelihood factor, the final (U,V) will be calculated by:

U = U_in + DU

V = V_in + DV

3.6.1.3 Adaptive Contrast Enhancement (ACE)

The Automatic Contrast Enhancement (ACE) is a part of the color processing pipe, which located at the render cache in the RCPB block.

The main goals of the ACE is to improve the overall contrast of the image, and emphasizing details when relevant (such as in dark areas).

The ACE algorithm analyzes the image, and consequently changes contrast of the image according to its characteristics. It works in YCbCr color space, where analysis and changes are performed over the Y component. The result of ACE is a 1d (1 dimension) look up table (1D LUT) operating on Y. The ACE follows the skin tone enhancement module in the pipe.

The ACE is receiving skin information from the STD block. When the frame includes skin the affect of the ACE is reduced in the skin area.

The ACE operation is divided into three stages:

1. Collecting information on Y and building the picture histogram. (Hardware)
2. Analysis on the collected data. (Software/Kernel)
3. Modification of the Y component. (Hardware)

The major steps of ACE can be divided into the following steps and depict in the below diagram.

1. Histogram calculation of the Y values.
2. Limiting extremely large histogram's bins.
3. Calculate the Image's gray level mean value (Ymean).
4. Calculate the Image's "Dark Factor" by the Ymean and external transfer function.
5. Find the PWLF anchor input and output points according to the "Portion Values" and the "Destination Points" of the Bright and the Dark images.
6. Find the PWLF anchor Input points by the blending of the Dark and Bright anchor input points, according to the Dark Factor calculated previously.
7. Find the PWLF anchor Output points by the blending of the Dark and Bright anchor output points, according to the Dark Factor calculated previously.
8. Limit Slopes between the anchor points. This stage's output is the current's image ACE PWLF.
9. "Soften" the ACE PWLF by blending I with the Identity Transformation.
10. Blend the current PWLF with the PWLF of the previous image (History blend).
11. Apply the final PWLF, and get the Yout values.

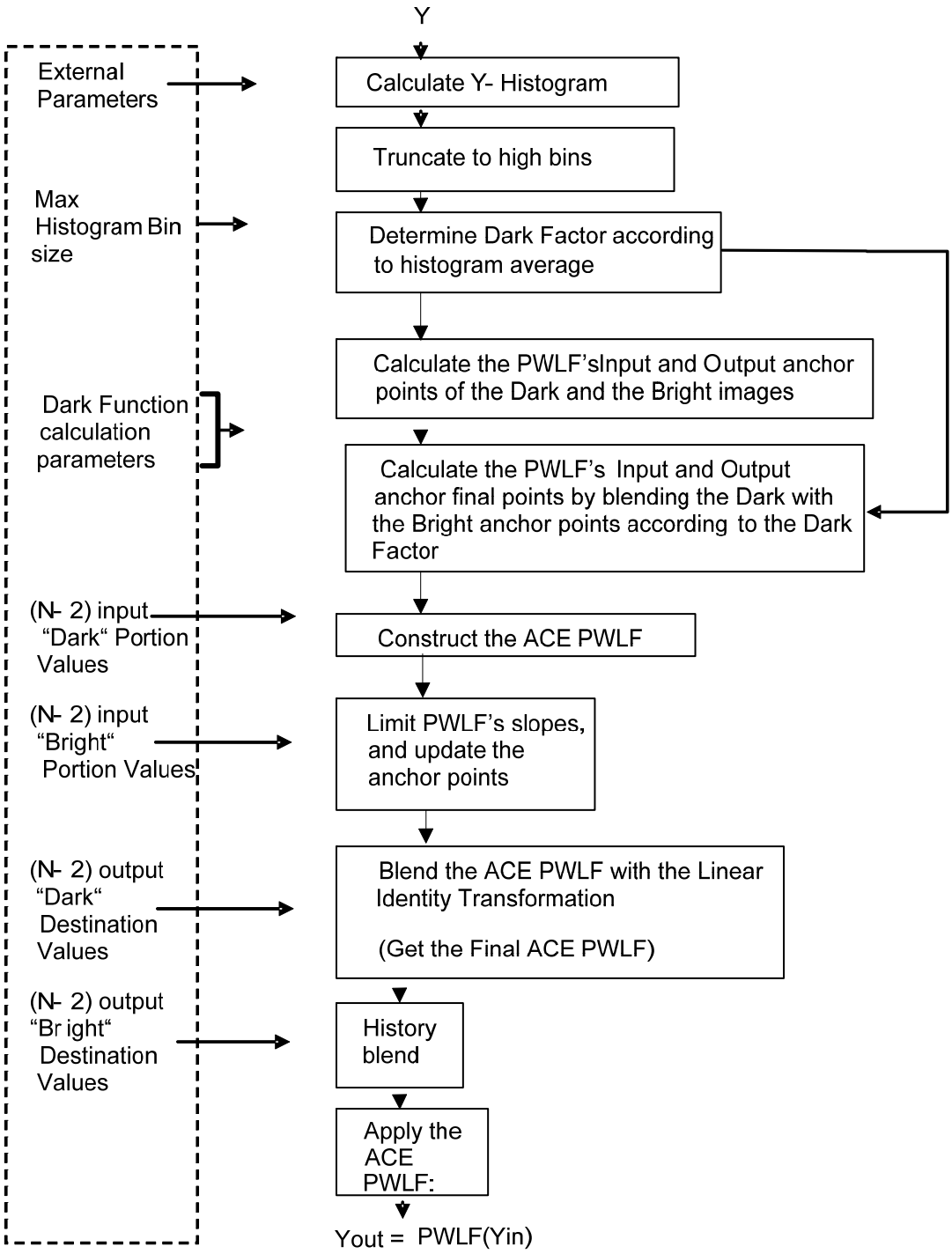
Note: Step 1 & step 11 are done in HW and steps 2-10 are done in software.

The main ACE goals are overall contrast improvement, and details emphasizing. ACE algorithm generates a Piece-wise Linear Function (PWLF), and the final gray values, Yout, are calculated by Yout = PWLF(Yin).

The HW compares the input pixels to the ***skin_threshold*** to determine if the target pixel is a skin pixel or not. It operates on all of the input pixels if the ***Full_image_histogram*** flag is defined. (to ignore the AOI flag). HW output the histogram of luma pixel value to VSC, and at VSC, the maximum and minimum value

of luma pixels (Ymax, Ymin) ans the number of skin pixels is determined to be made available to the software development via MMIO register.

An eleven-segment (12 points) was established to implement PWLF via the state parameters (Points: ***Ymin, Y1-Y10, Ymax***, Bias: ***B1 – B10***, Slope: ***S0-S10***).


Y
External Parameters
Calculate Y- Histogram
Truncate to high bins
Max Histogram Bin size
Determine Dark Factor according to histogram average
Calculate the PWLF'sInput and Output anchor points of the Dark and the Bright images
Dark Function calculation parameters
Calculate the PWLF's Input and Output anchor final points by blending the Dark with the Bright anchor points according to the Dark Factor
(N- 2) input "Dark" Portion Values
Construct the ACE PWLF
(N- 2) input "Bright" Portion Values
Limit PWLF's slopes, and update the anchor points
(N- 2) output "Dark" Destination Values
Blend the ACE PWLF with the Linear Identity Transformation
(Get the Final ACE PWLF)
(N- 2) output "Br ight" Destination Values
History blend
Apply the ACE PWLF:
Yout = PWLF(Yin)

3.6.1.4 Total Color Control (TCC)

The TCC allows users to choose different grades of saturation for each of the six basic colors (Red, Green, Blue, Magenta, Yellow and Cyan) in order to custom the color scheme. The TCC algorithm operates on the UV-color components in the YUV color space. It operates in the pixel-wise mode, without considering any neighborhood information.

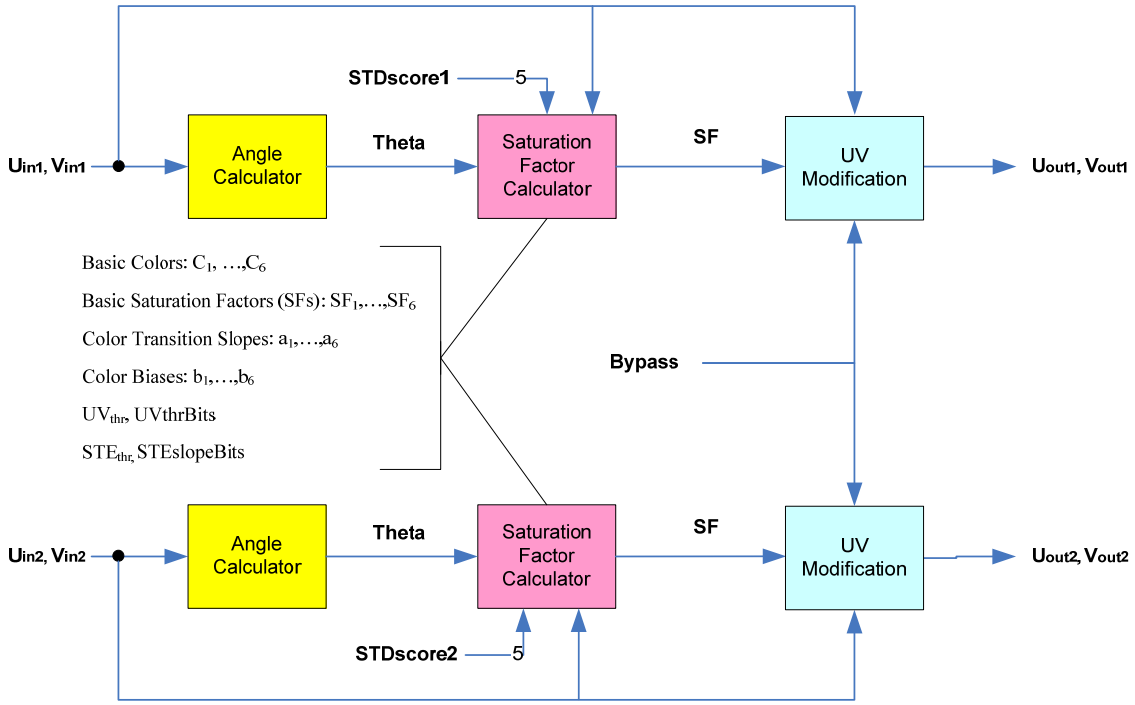
Its input is:

1. U,V color components (10 bit)
2. Skin-tone detection value (5 bit)
3. External control parameters

Its output is the new U, V values (10 bit).

The motivation to implement this block in HW is to reduce the power of the system and therefore the battery life.

The pixel TPT (throughput) is two pixels per clock. The pipeline works in YUV formats only – 10bit pixels. The TCC block is control by state only and does not require any memory access. The TCC block runs at the same frequency of the existing RCPBunit.



There are two paths in parallel to support the requirement of two pixels per clock. Valid out is a signal which high when the pixels are valid.

The TCC block includes three sub blocks.

Angle_calculator

This block receive pixel U and V and perform division of abs|v| by abs|u| using Divider ROM with pipeline.

The division result is used to calculated arctan of the V/U. This result is used to calculate the angle called θ, by using approximation equation. This angle is defined as a 10bit.

To simplify this calculation the "arctangent" function is approximated in the $[0,45]^o$ region by the second order polynomial:

$$\theta = \arctan(x) = -0.2880x^2 + 1.0797x - 0.005; \quad (0 <= x <= 1)$$

The resulted θ is given in radians with the maximal error of 0.005 rad. (0.286 deg.) This approximation is calculated by the minimizing the mean squared error (mse) between the actual "arctan" function, and its polynomial approximation, and thus represents the optimal mse-approximation in the [0,π/4] region. The θ for the all regions is calculated by:

$$\theta = \begin{cases} \theta_{0.25\pi}; & \text{for region I, } (0 <= x <= 1), \\ \pi/2 - \theta_{0.25\pi}; & \text{for region II, } (1 < (V/U) < \text{infinity}) \\ \pi/2 + \theta_{0.25\pi}; & \text{for region III, } (-\text{infinity} < (V/U) < -1) \\ \pi - \theta_{0.25\pi}; & \text{for region IV, } (-1 <= (V/U) < 0) \\ \pi + \theta_{0.25\pi}; & \text{for region V, } (0 <= (V/U) <= 1) \\ 3\pi/2 - \theta_{0.25\pi}; & \text{for region VI, } (1 < (V/U) < \text{infinity}) \\ 3\pi/2 + \theta_{0.25\pi}; & \text{for region VII, } (-\text{infinity} < (V/U) < -1) \\ 2\pi - \theta_{0.25\pi}; & \text{for region VIII, } (-1 <= (V/U) < 0) \end{cases}$$

Whereas x = (V/U), and the $\theta_{0.25\pi}$ is given by the above equation.

Saturation_Factor_Calculator

This block is using the angle θ, locate where it is in the color wheel, find the appropriate base colors and calculate the proportional distance from the adjacent base color. The result called α . Alpha (α) represent the distance from the two relevant base color.

Calculate the saturation by using the appropriate user parameters. The result is the Saturation factor. This block considering also the threshold and the maximum UV values, and considering also correction for gray colors to minimize the possible noise. In addition the saturation skipping doing saturation when the color is skin and doing alpha blending according the skin factor called STDscore.

This block requires several external parameters such:

BaseColor1,…, BaseColor6 – Six basic user defined colors.

SatFactor1,…, SatFactor6 – Six basic saturation change user defined factors.

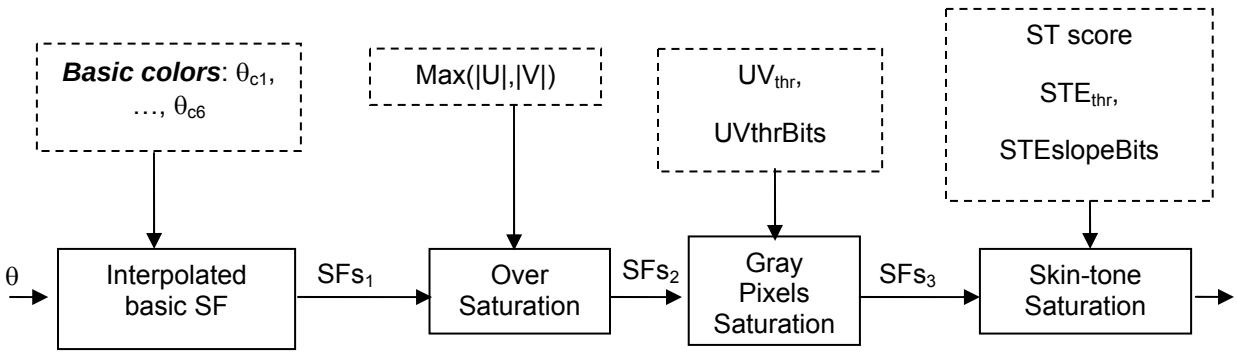
ColorTransitSlope12, …..ColorTransit61 – Six calculation result of 1/(BaseColorX – BaseColorY)

ColorBias1,…, ColorBias6 – Six color bias.

STDscore – Skin-tone Detection score (from STD/E).

The result of SF is a number of 8bits.

There are four major steps to derive the saturation factor.



Calculation of the Saturation Factor (SF)

θ – current pixel's color as calculated by the Eq. (3)

Lined boxes show additional data used by each block.

SFs_i – SF after the step "i".

SFs_4 is the SF_{final}.

The Interpolated Basic SFs_1

With the calculated angle θ, which lies in the $[\theta_{Ci}, \theta_{Ci+1}]$ interval, the Interpolated Basic SFs_1 will be:

$SFs_1 = (1-\alpha)\ \boldsymbol{SatFactor_i} + \alpha\ \boldsymbol{SatFactor_{i+1}}$

Whereas α is calculated by:

$$\alpha = \mathrm{Min}\{\mathrm{Max}[(\theta - \boldsymbol{BaseColor_i})*\boldsymbol{ColorTransitSlope_i} - \boldsymbol{ColorBias_i}, 0], 1\}$$

Over Saturation Limiter SFs2

Over Saturation Limiter block is used to avoid saturation boosting of the already high saturated pixels. The SFs_2 is calculated by:

$$SFs_2 = \begin{cases} SFs_1, & \text{for } (SF_1 <= 1) \\ 1 + (SFs_1 - 1)(\mathrm{MaxColor} - UV_{max})/\mathrm{MaxColor}, & \text{for } (1 < SF_1 <= 2) \text{ AND } (UV_{max} <= \boldsymbol{UVMaxColor}) \\ 1, & \text{for } (UV_{max} > \boldsymbol{UVMaxColor}) \end{cases}$$

Where the UV_{max} = max(|U|,|V|), and ***UVMaxColor*** is an external parameter which in the case of YUV color space is equal to 448 in 10bit representation. ***The Inv_UVMaxColor*** was used for the inverse calculation of 1/UVMaxColor.

Note: *The last condition (UV_{max} > **UVMaxColor**) is associated with the illegal colors, and usually hasn't to appear.*

GrayPixels Saturation LimiterSFs3

This block limits the saturation of the almost gray pixels. The reason for this limiter is to prevent the noise amplification by the Saturation increase process. The result of this block is:

$$SFs_3 = 1 + dSF * CLF$$

Where:

$$dSF = SFs_2 - 1;$$

And the CLF is called Color Limiting Factor and ranges from 0 to 1. The calculation of the CLF is given by:

$$CLF = \begin{cases} 1; & \text{for } (SFs_2 <= 1) \text{ AND (any } UV_{max}) \\ 0; & \text{for } (UV_{max} <= \textbf{UV_Threshold}) \\ (UV_{max} - \textbf{UV_Threshold}) / 2^{UV_Threshold_Bits}; & \text{for } (\textbf{UV_Threshold} < UV_{max} < (\textbf{UV_Threshold} + 2^{UV_Threshold_Bits})) \end{cases}$$

Skin-tone Saturation LimiterSFs4

The last block effects TCC strength operation of the Skin-tone pixels. Uncontrolled enhancement of the skin pixels could lead to appearing of artifacts and to undesired results. The final SFs_4 is calculated by a linear blending:

$$SFs_4 = (128*STE_{factor} + (256 - STE_{factor})\, SFs_3) / 256$$

Where the STE_{factor} is called Skin Tone factor and is calculated by:

$$diff = (STD_{score} - \textbf{STE_Threshold}) * 2^3$$

*Note: the STD_{score} (from STD) and the **STE_Threshold** are presented with 5 bits. The multiplication by 2^3 is in order to raise the "diff" to 8 bits.*

$$STE_{factor} = Min\{Max[(diff * 2^{STE_SlopeBits}), 0], 255\}$$

The STD_{score} is a result of the Skin-tone Detection module. It is represented with 5 bits, where the values 0 and 31 mean no skin-tone, and full skin-tone detection, respectively. The STE_{factor} is given by 8 bits, where the value 256 represents the number 1.

It is evident that for the high values of STE_{factor} the resulted SFs_4 is close to 1, which means a weak TCC action of this pixel (SFs_4 = 1 actually means TCC is off).

UV Modification – The input pixels are multiple by the saturation factor. The results are the output pixels.

SF_{final} is the final saturation factor which actually resulted from the forth SFcalculation block:

$$SF_{final} = SFs_4$$

The calculation of the U_{new}, and V_{new} output values. They are calculated below:

$$U_{new} = U * SF_{final}$$

$$V_{new} = V * SF_{final}$$

Whereas (U,V) are the original input color components,

Because these pixels are represented in the unbiased form, which is the result of substraction of the value 512 from the original [U,V] values, the final [U_{out}, V_{out}] values are given by:

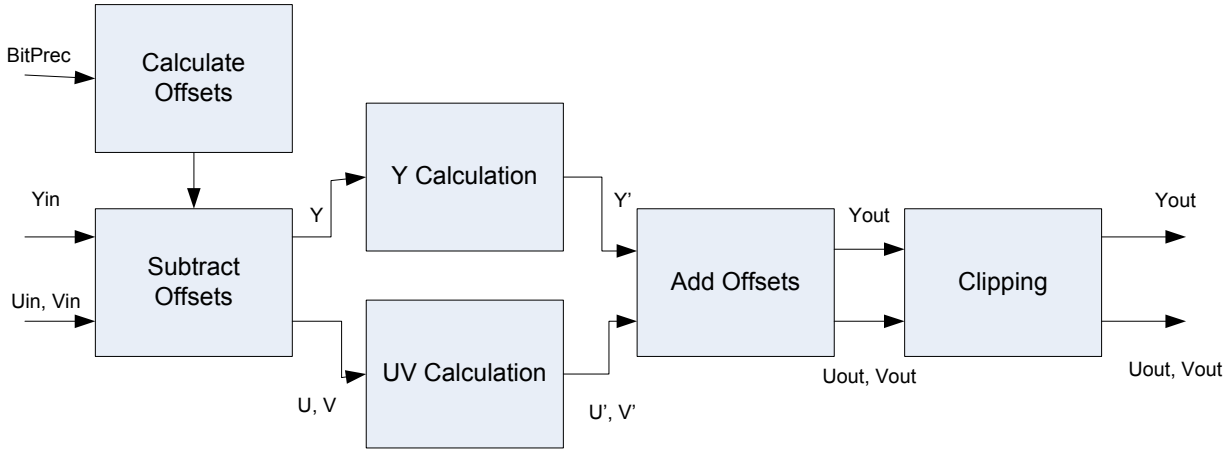
$$U_{out} = U_{new} + 512$$

$$V_{out} = V_{new} + 512$$

This is the final TCC output represented with 10 bits.

3.6.1.5 ProcAmp

The PROCAMP block modifies the brightness, contrast, hue and saturation of an image in YCbCr color space (or similar).



The algorithm itself uses 8-16 bits per color.

Y Processing: 256 is subtracted from the Y values to position the black level at zero. This removes the DC offset so that adjusting the contrast does not vary the black level. Since Y values may be less than 256, negative Y values should be supported at this point. Contrast is adjusted by multiplying the YUV pixel values by a constant. If U and V are adjusted, a color shift will result whenever the contrast is changed. The brightness property value is added (or subtracted) from the contrast adjusted Y values; this is done to avoid introducing a DC offset due to adjusting the contrast. Finally the value 64 is added to reposition the black level at 256. The equation for processing of Y values is:

Y' = ((Y-256) x C) + B + 256,

where C is the ***Contrast*** value and B is the ***Brightness*** value.

UV Processing: 2048 is first subtracted from both U and V values to position the range around zero. The hue property is implemented by mixing the U and V values together:

U' = (U-2048) x Cos(H) + (V-2048) x Sin(H)

V' = (V-2048) x Cos(H) – (U-2048) x Sin(H)

Where H represents the desired Hue angle; Saturation is adjusted by multiplying both U and V by a constant.

Finally, the value 2048 is added to both U and V. The combined processing of Hue and Saturation on the UV data is:

U' = (((U-2048) x Cos(H) + (V-2048) x Sin(H)) x C x S) + 2048

V' = (((V-2048) x Cos(H) - (U-2048) x Sin(H)) x C x S) + 2048

Where C is the contrast, H is Hue angle and S is the Saturation and the combination of Cos(H)*C*S and Sin(H)*C*S is specified by parameters ***Cos_c_s*** and ***Sin_c_s.***

3.6.1.6 Color Space Conversion

The CSC block enables linear conversion between color spaces using vector shift, matrix multiplication, and additional shift.

The CSC algorithm is a linear coordinate transformation, comprising of the following stages:

- Shifting the input color coordinate.
- Multiply by 3*3 matrix
- Shifting the output color coordinate
- Formula representation of last 3 steps:

$$\begin{pmatrix} \text{vout_1} \\ \text{vout_2} \\ \text{vout_3} \end{pmatrix} = \begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{pmatrix} * \begin{pmatrix} \text{vin_1+v0_1} \\ \text{vin_2+v0_2} \\ \text{vin_3+v0_3} \end{pmatrix} + \begin{pmatrix} \text{u0_1} \\ \text{u0_2} \\ \text{u0_3} \end{pmatrix}$$

Where:

aij are the matrix elements, i.e., the transform coefficients: ***C0, C1, C2, C3, C4, C5, C6, C7, C8.***

vin_i is the input pixel color components

v0_i is the input offset vector, i.e., ***Offset_in_1***, ***Offset_in_2, Offset_in_3.***

u0_1_i is the output offset vector. i.e., ***Offset_out_1, Offset_out_2, Offset_out_3.***

Clipping the output to ensure each component is in allowed range.

The parameters ***YUV_IN*** is used to set input to be RGB format and ***YUV_OUT*** is uased to set output to be RGB format

Notes about Repacker:

There are two states to be used in the repacker: ***Alpha from State Select*** and ***color pipe alpha***. The last module in the IECP pipeline.

If Alpha from State Select is set, the Y, U ,V is packed with the information from color pipe alpha, and then the data is sent out to RCPB.

Otherwise, "0" is inserted in the 4LSB (alpha) and the packed data is sent out to RCPB.

3.7 Accessing Render Targets

Render targets are the surfaces that the final results of pixel shaders are written to. The render targets support a large set of surface formats (refer to surface formats table in *Sampling Engine* for details) with hardware conversion from the format delivered by the thread. The render target message also causes numerous side effects, including potentially alpha test, depth test, stencil test, alpha blend (which normally causes a read of the render target), and other functions. These functions are covered in the *Windower* chapter as some of them (depth/stencil test) are also partially done in the Windower.

The render target write messages are specifically for the use of pixel shader threads that are spawned by the windower, and may not be used by any other threads. This is due to the pixel scoreboard side-effects that sending of this message entails. The pixel scoreboard ensures that incorrect ordering of reads and writes to the same pixel does not occur.

3.7.1 Single Source

The "normal" render target messages are single source. There are two forms, SIMD16 and SIMD8, intended for the equivalent-sized pixel shader threads. A single color (4 channels) is delivered for each of the 16 or 8 pixels in the message payload. Optional depth, stencil, and antialias alpha information can also be delivered with these messages.

The pixel scoreboard bits corresponding to the dispatched pixel mask (or half of the mask in the case of SIMD8 messages) are cleared only if the **Last Render Target Select** bit is set in the message descriptor.

The single source message will not cause a write to the render target if **Dual Source Blend Enable** in 3DSTATE_WM is *enabled*. However, if **Last Render Target Select** is set, the message will still cause pixel scoreboard clear and depth/stencil buffer updates if enabled.

3.7.2 Dual Source [DevSNB+]

The dual source render target messages only have SIMD8 forms due to maximum message length limitations. SIMD16 pixel shaders must send two of these messages to cover all of the pixels. Each message contains two colors (4 channels each) for each pixel in the message payload. In addition to the first source, the second source can be selected as a blend factor (BLENDFACTOR_*_SRC1_* options in the blend factor fields of COLOR_CALC_STATE or BLEND_STATE). Optional depth, stencil, and antialias alpha information can also be delivered with these messages.

Each dual source message delivered will clear the corresponding pixel scoreboard bits if the **Last Render Target Select** bit in the message descriptor is set.

The dual source message will revert to a single source message using source 0 if **Dual Source Blend Enable** in 3DSTATE_WM is disabled.

3.7.3 Replicate Data

The replicate data render target message is intended to be used for "fast clear" functionality in cases where the color data for each pixel is identical. This message performs better than the other messages due to its smaller message length. This message does not support depth, stencil, or antialias alpha data being sent with it. This message must target only tiled memory. Access of linear memory using this message type is UNDEFINED. The depth buffer can be cleared through the "early depth" function in

conjunction with a pixel shader using this message. Refer to the *Windower* chapter for more details on the early depth function.

The pixel scoreboard bits corresponding to the dispatched pixel mask are cleared only if the **Last Render Target Select** bit is set in the message descriptor.

3.7.4 Multiple Render Targets (MRT)

Multiple render targets are supported with the single source and replicate data messages. Each render target is accessed with a separate Render Target Write message, each with a different surface indicated (different binding table index). The depth buffer is written only by the message(s) to the last render target, indicated by the **Last Render Target Select** bit set to clear the pixel scoreboard bits.

[DevSNB+]: MRT is not supported when one or more RTs have this surface formats: YCRCB_SWAPUVY, YCRCB_SWAPUV, YCRCB_SWAPY, YCRCB_NORMAL

3.8 State

3.8.1 BINDING_TABLE_STATE

The data port uses the binding table to retrieve surface state. Refer to *Sampling Engine* for the definition of this state.

3.8.2 SURFACE_STATE

The data port uses the surface state for constant buffers, render targets, and media surfaces. Refer to *Sampling Engine* for the definition of this state.

3.8.3 COLOR_PROCESSING_STATE [DevSNB+]

This state structure contains the state used by the color processing function.

DWord	Bit	Description
		STD / STE State
0	31:24	**V_Mid:** Rectangle middle-point V coordinate. Format = U8 (The default is 154)
	23:16	**U_Mid:** Rectangle middle-point U coordinate. Format = U8 (The default is 110)
	15:10	**Hue_Max:** Rectangle half width. Format = U6 (The default is 14)
	9:4	**Sat_Max:** Rectangle half length. Format = U6 (The default is 31)

DWord	Bit	Description
	3	Reserved : MBZ
	2	**Output Control** 0: Output Pixels 1: Output STD Decisions
	1	**STE Enable** Format = Enable
	0	**STD Enable** Format = Enable
1	31	Reserved : MBZ
	30:28	**Diamond Margin** Format = U3 (The default is 4)
	27:21	**Diamond_du:** Rhombus center shift in the sat-direction, relative to the rectangle center. Format = S7 2's complement (The default is 0)
	20:18	**HS_margin:** Defines rectangle margin. Format = U3 (The default is 3)
	17:10	**Cos(α)** Format = S0.7 2's complement (The default is 79/128)
	9:8	**Reserved: MBZ**
	7:0	**Sin(α)** Format = S0.7 2's complement (The default is 101/128)
2	31:21	Reserved : MBZ
	20:13	**Diamond_alpha:** 1 / tan(β) Format = U2.6 (The default is 100/64)
	12:7	**Diamond_Th:** Half length of the rhombus axis in the sat-direction. Format = U6 (The default is 35)
	6:0	**Diamond_dv:** Rhombus center shift in the hue-direction, relative to the rectangle center. Format = S6 2's complement (The default is 0)
3	31:24	**Y_point_3:** Third point of the Y piecewise linear membership function. Format = U8 (The default is 254)
	23:16	**Y_point_2:** Second point of the Y piecewise linear membership function. Format = U8 (The default is 47)
	15:8	**Y_point_1:** First point of the Y piecewise linear membership function. Format = U8 (The default is 46)
	7	**VY_STD_Enable:** Enables STD in the VY subspace. Format = Enable
	6:0	Reserved : MBZ

DWord	Bit	Description
4	31:18	Reserved : MBZ
	17:13	**Y_Slope_2:** Slope between points Y3 and Y4. Format = U2.3 (The default is 31/8)
	12:8	**Y_Slope_1:** Slope between points Y1 and Y2. Format = U2.3 (The default is 31/8)
	7:0	**Y_point_4:** Fourth point of the Y piecewise linear membership function. Format = U8 (The default is 255)
5	31:16	**INV_skin_types_margin**: 1/(2* Skin_types_margin) Format = U0.16 (Skin_Type_margin = 20)
	15:0	**INV_Margin_VYL:** 1 / Margin_VYL Format = U0.16 (Margin_VYL = 6554/65536)
6	31:24	**P1L:** Y Point 1 of the lower part of the detection PWLF. Format = U8 (The default is 216)
	23:16	**P0L:** Y Point 0 of the lower part of the detection PWLF. Format = U8 (The default is 46)
	15:0	**INV_Margin_VYU:** 1 / Margin_VYU Format = 0.16 (Margin_VYU = 3227/65536)
7	31:24	**B1L:** V Bias 1 of the lower part of the detection PWLF. Format = U8 (The default is 130)
	23:16	**B0L:** V Bias 0 of the lower part of the detection PWLF. Format = U8 (The default is 133)
	15:8	**P3L:** Y Point 3 of the lower part of the detection PWLF. Format = U8 (The default is 236)
	7:0	**P2L:** Y Point 2 of the lower part of the detection PWLF. Format = U8 (The default is 236)
8	31:27	Reserved : MBZ
	26:16	**S0L:** Slope 0 of the lower part of the detection PWLF. Format = S2.8 2's complement (The default is -5/256)
	15:8	**B3L:** V Bias 3 of the lower part of the detection PWLF. Format = U8 (The default is 130)
	7:0	**B2L:** V Bias 2 of the lower part of the detection PWLF. Format = U8 (The default is 130)
9	31:22	Reserved : MBZ
	21:11	**S2L:** Slope 2 of the lower part of the detection PWLF. Format = S2.8 2's complement (The default is 0/256)

DWord	Bit	Description
	10:0	**S1L:** Slope 1 of the lower part of the detection PWLF. Format = S2.8 2's complement (The default is 0/256)
10	31:27	Reserved : MBZ
	26:19	**P1U:** Y Point 1 of the upper part of the detection PWLF. Format = U8 (The default is 66)
	18:11	**P0U:** Y Point 0 of the upper part of the detection PWLF. Format = U8 (The default is 46)
	10:0	**S3L:** Slope 3 of the lower part of the detection PWLF. Format = S2.8 2's complement (The default is 0/256)
11	31:24	**B1U:** V Bias 1 of the upper part of the detection PWLF. Format = U8 (The default is 163)
	23:16	**B0U:** V Bias 0 of the upper part of the detection PWLF. Format = U8 (The default is 143)
	15:8	**P3U:** Y Point 3 of the upper part of the detection PWLF. Format = U8 (The default is 236)
	7:0	**P2U:** Y Point 2 of the upper part of the detection PWLF. Format = U8 (The default is 150)
12	31:27	Reserved : MBZ
	26:16	**S0U:** Slope 0 of the upper part of the detection PWLF. Format = S2.8 2's complement (The default is 256/256)
	15:8	**B3U:** V Bias 3 of the upper part of the detection PWLF. Format = U8 (The default is 140)
	7:0	**B2U:** V Bias 2 of the upper part of the detection PWLF. Format = U8 (The default is 200)
13	31:22	Reserved : MBZ
	21:11	**S2U:** Slope 2 of the upper part of the detection PWLF. Format = S2.8 2's complement (The default is -179/256)
	10:0	**S1U:** Slope 1 of the upper part of the detection PWLF. Format = S2.8 2's complement (The default is 113/256)
14	31:28	Reserved : MBZ
	27:20	**Skin_types_margin:** Skin types Y margin Format = U8 (The default is 20)
	19:12	**Skin_types_thresh:** Skin types Y threshold Format = U8 (The default is 120)
	11	**Skin_types_enable:** Treat differently bright and dark skin types Format = Enable

DWord	Bit	Description
	10:0	**S3U:** Slope 3 of the upper part of the detection PWLF. Format = S2.8 2's complement (The default is 0/256)
15	31	Reserved : MBZ
	30:21	**SATB1:** First bias for the saturation PWLF (bright skin). Format = S7.2 2's complement (The default is 0/4)
	20:14	**SATP3:** Third point for the saturation PWLF (bright skin). Format = S6 2's complement (The default is 31)
	13:7	**SATP2:** Second point for the saturation PWLF (bright skin). Format = S6 2's complement (The default is 31)
	6:0	**SATP1:** First point for the saturation PWLF (bright skin). Format = S6 2's complement (The default is -11)
16	31	Reserved : MBZ
	30:20	**SATS0:** Zeroth slope for the saturation PWLF (bright skin) Format = U3.8 (The default is 397/256)
	19:10	**SATB3:** Third bias for the saturation PWLF (bright skin) Format = S7.2 2's complement (The default is 124/4)
	9:0	**SATB2:** Second bias for the saturation PWLF (bright skin) Format = S7.2 2's complement (The default is 124/4)
17	31:22	Reserved : MBZ
	21:11	**SATS2:** Second slope for the saturation PWLF (bright skin) Format = U3.8 (The default is 256/256)
	10:0	**SATS1:** First slope for the saturation PWLF (bright skin) Format = U3.8 (The default is 189/256)
18	31:25	**HUEP3:** Third point for the hue PWLF (bright skin) Format = S6 2's complement (The default is 14)
	24:18	**HUEP2:** Second point for the hue PWLF (bright skin) Format = S6 2's complement (The default is 2)
	17:11	**HUEP1:** First point for the hue PWLF (bright skin) Format = S6 2's complement (The default is 0)
	10:0	**SATS3:** Third slope for the saturation PWLF (bright skin) Format = U3.8 (The default is 256/256)
19	31:30	Reserved : MBZ
	29:20	**HUEB3:** Third bias for the hue PWLF (bright skin) Format = S7.2 2's complement (The default is 56/4)
	19:10	**HUEB2:** Second bias for the hue PWLF (bright skin) Format = S7.2 2's complement (The default is 0/4)

DWord	Bit	Description
	9:0	**HUEB1:** First bias for the hue PWLF (bright skin) Format = S7.2 2's complement (The default is 0/4)
20	31:22	Reserved : MBZ
	21:11	**HUES1:** First slope for the hue PWLF (bright skin) Format = U3.8 (The default is 0/256)
	10:0	**HUES0:** Zeroth slope for the hue PWLF (bright skin) Format = U3.8 (The default is 256/256)
21	31:22	Reserved : MBZ
	21:11	**HUES3:** Third slope for the hue PWLF (bright skin) Format = U3.8 (The default is 256/256)
	10:0	**HUES2:** Second slope for the hue PWLF (bright skin) Format = U3.8 (The default is 299/256)
22	31	Reserved : MBZ
	30:21	**SATB1_DARK:** First bias for the saturation PWLF (dark skin) Format = S7.2 2's complement (The default is 0/4)
	20:14	**SATP3_DARK:** Third point for the saturation PWLF (dark skin) Format = S6 2's complement (The default is 31)
	13:7	**SATP2_DARK:** Second point for the saturation PWLF (dark skin) Format = S6 2's complement (The default is 31)
	6:0	**SATP1_DARK:** First point for the saturation PWLF (dark skin) Format = S6 2's complement (The default is -11)
23	31	Reserved : MBZ
	30:20	**SATS0_DARK:** Zeroth slope for the saturation PWLF (dark skin) Format = U3.8 (The default is 397/256)
	19:10	**SATB3_DARK:** Third bias for the saturation PWLF (dark skin) Format = S7.2 2's complement (The default is 124/4)
	9:0	**SATB2_DARK:** Second bias for the saturation PWLF (dark skin) Format = S7.2 2's complement (The default is 124/4)
24	31:22	Reserved : MBZ
	21:11	**SATS2_DARK:** Second slope for the saturation PWLF (dark skin) Format = U3.8 (The default is 256/256)
	10:0	**SATS1_DARK:** First slope for the saturation PWLF (dark skin) Format = U3.8 (The default is 189/256)
25	31:25	**HUEP3_DARK:** Third point for the hue PWLF (dark skin). Format = S6 2's complement (The default is 14)

DWord	Bit	Description
	24:18	**HUEP2_DARK:** Second point for the hue PWLF (dark skin). Format = S6 2's complement (The default is 2)
	17:11	**HUEP1_DARK:** First point for the hue PWLF (dark skin). Format = S6 2's complement (The default is 0)
	10:0	**SATS3_DARK:** Third slope for the saturation PWLF (dark skin) Format = U3.8 (The default is 256/256)
26	31:30	Reserved : MBZ
	29:20	**HUEB3_DARK:** Third bias for the hue PWLF (dark skin). Format = S7.2 2's complement (The default is 56/4)
	19:10	**HUEB2_DARK:** Second bias for the hue PWLF (dark skin). Format = S7.2 2's complement (The default is 0/4)
	9:0	**HUEB1_DARK:** First bias for the hue PWLF (dark skin). Format = S7.2 2's complement (The default is 0/4)
27	31:22	Reserved : MBZ
	21:11	**HUES1_DARK:** First slope for the hue PWLF (dark skin). Format = U3.8 (The default is 0/256)
	10:0	**HUES0_DARK:** Zeroth slope for the hue PWLF (dark skin). Format = U3.8 (The default is 256/256)
28	31:22	Reserved : MBZ
	21:11	**HUES3_DARK:** Third slope for the hue PWLF (dark skin). Format = U3.8 (The default is 256/256)
	10:0	**HUES2_DARK:** Second slope for the hue PWLF (dark skin). Format = U3.8 (The default is 299/256)
		ACE State
29	31:7	Reserved : MBZ
	6:2	**Skin_threshold:** Used for Y analysis (min/max) for pixels which are higher than skin threshold. Format = U5 (The default is 26)
	1	**Full_image_histogram:** Used to ignore the area of interest for full image histogram. Format = Enable (The default is 0)
	0	**ACE Enable** Format = Enable
30	31:24	**Y3:** The value of the y_pixel for point 3 in PWL. Format = U8 (The default is 76)
	23:16	**Y2:** The value of the y_pixel for point 2 in PWL. Format = U8 (The default is 56)

DWord	Bit	Description
	15:8	**Y1:** The value of the y_pixel for point 1 in PWL. Format = U8 (The default is 36)
	7:0	**Ymin:** The value of the y_pixel for point 0 in PWL. Format = U8 (The default is 16)
31	31:24	**Y7:** The value of the y_pixel for point 7 in PWL. Format = U8 (The default is 156)
	23:16	**Y6:** The value of the y_pixel for point 6 in PWL. Format = U8 (The default is 136)
	15:8	**Y5:** The value of the y_pixel for point 5 in PWL. Format = U8 (The default is 116)
	7:0	**Y4:** The value of the y_pixel for point 4 in PWL. Format = U8 (The default is 96)
32	31:24	**Ymax:** The value of the y_pixel for point 11 in PWL. Format = U8 (The default is 235)
	23:16	**Y10:** The value of the y_pixel for point 10 in PWL. Format = U8 (The default is 216)
	15:8	**Y9:** The value of the y_pixel for point 9 in PWL. Format = U8 (The default is 196)
	7:0	**Y8:** The value of the y_pixel for point 8 in PWL. Format = U8 (The default is 176)
33	31:24	**B4:** The value of the bias for point 4 in PWL. Format = U8 (The default is 96)
	23:16	**B3:** The value of the bias for point 3 in PWL. Format = U8 (The default is 76)
	15:8	**B2:** The value of the bias for point 2 in PWL. Format = U8 (The default is 56)
	7:0	**B1:** The value of the bias for point 1 in PWL. Format = U8 (The default is 36)
34	31:24	**B8:** The value of the bias for point 8 in PWL. Format = U8 (The default is 176)
	23:16	**B7:** The value of the bias for point 7 in PWL. Format = U8 (The default is 156)
	15:8	**B6:** The value of the bias for point 6 in PWL. Format = U8 (The default is 136)
	7:0	**B5:** The value of the bias for point 5 in PWL. Format = U8 (The default is 116)

DWord	Bit	Description
35	31:16	Reserved : MBZ
	15:8	**B10:** The value of the bias for point 10 in PWL. Format = U8 (The default is 216)
	7:0	**B9:** The value of the bias for point 9 in PWL. Format = U8 (The default is 196)
36	31:27	Reserved : MBZ
	26:16	**S1:** The value of the slope for point 1 in PWL Format = U1.10 (The default is 1024/1024)
	15:11	Reserved : MBZ
	10:0	**S0:** The value of the slope for point 0 in PWL Format = U1.10 (The default is 1024/1024)
37	31:27	Reserved : MBZ
	26:16	**S3:** The value of the slope for point 3 in PWL Format = U1.10 (The default is 1024/1024)
	15:11	Reserved : MBZ
	10:0	**S2:** The value of the slope for point 2 in PWL Format = U1.10 (The default is 1024/1024)
38	31:27	Reserved : MBZ
	26:16	**S5:** The value of the slope for point 5 in PWL Format = U1.10 (The default is 1024/1024)
	15:11	Reserved : MBZ
	10:0	**S4:** The value of the slope for point 4 in PWL Format = U1.10 (The default is 1024/1024)
39	31:27	Reserved : MBZ
	26:16	**S7:** The value of the slope for point 7 in PWL Format = U1.10 (The default is 1024/1024)
	15:11	Reserved : MBZ
	10:0	**S6:** The value of the slope for point 6 in PWL Format = U1.10 (The default is 1024/1024)
40	31:27	Reserved : MBZ
	26:16	**S9:** The value of the slope for point 9 in PWL Format = U1.10 (The default is 1024/1024)
	15:11	Reserved : MBZ
	10:0	**S8:** The value of the slope for point 8 in PWL Format = U1.10 (The default is 1024/1024)

DWord	Bit	Description
41	31:11	Reserved : MBZ
	10:0	**S10:** The value of the slope for point 10 in PWL Format = U1.10 (The default is 1024/1024)
		TCC State
42	31:24	**SatFactor3:** The saturation factor for yellow. Format = U1.7 (The default is 220)
	23:16	**SatFactor2:** The saturation factor for red. Format = U1.7 (The default is 220)
	15:8	**SatFactor1:** The saturation factor for magenta. Format = U1.7 (The default is 220)
	7	**TCC Enable** Format = Enable
	6:0	Reserved : MBZ
43	31:24	**SatFactor6:** The saturation factor for blue. Format = U1.7 (The default is 220)
	23:16	**SatFactor5:** The saturation factor for cyan. Format = U1.7 (The default is 220)
	15:8	**SatFactor4:** The saturation factor for green. Format = U1.7 (The default is 220)
	7:0	Reserved : MBZ
44	31:30	Reserved : MBZ
	29:20	**BaseColor3:** Base Color 3 Format = U10 (The default is 483)
	19:10	**BaseColor2:** Base Color 2 Format = U10 (The default is 307)
	9:0	**BaseColor1:** Base Color 1 Format = U10 (The default is 145)
45	31:30	Reserved : MBZ
	29:20	**BaseColor6:** Base Color 6 Format = U10 (The default is 995)
	19:10	**BaseColor5:** Base Color 5 Format = U10 (The default is 819)
	9:0	**BaseColor4:** Base Color 4 Format = U10 (The default is 657)
46	31:16	**ColorTransitSlope23:** The calculation result of 1 / (BC3 – BC2) [1/62] Format = U0.16 (The default is 744)

DWord	Bit	Description
	15:0	**ColorTransitSlope12:** The calculation result of 1 / (BC2 – BC1) [1/57] Format = U0.16 (The default is 405)
47	31:16	**ColorTransitSlope45:** The calculation result of 1 / (BC5 – BC4) [1/57] Format = U0.16 (The default is 407)
		ColorTransitSlope34: The calculation result of 1 / (BC4 – BC3) [1/61] Format = U0.16 (The default is 1131)
48	31:16	**ColorTransitSlope61:** The calculation result of 1 / (BC1 – BC6) [1/62] Format = U0.16 (The default is 377)
	15:0	**ColorTransitSlope56:** The calculation result of 1 / (BC6 – BC5) [1/62] Format = U0.16 (The default is 372)
49	31:22	**ColorBias3:** Color bias for BaseColor3. Format = U2.8 (The default is 0)
	21:12	**ColorBias2:** Color bias for BaseColor2. Format = U2.8 (The default is 150)
	11:2	**ColorBias1:** Color bias for BaseColor1. Format = U2.8 (The default is 0)
	1:0	Reserved : MBZ
50	31:22	**ColorBias6:** Color bias for BaseColor6. Format = U2.8 (The default is 0)
	21:12	**ColorBias5:** Color bias for BaseColor5. Format = U2.8 (The default is 0)
	11:2	**ColorBias4:** Color bias for BaseColor4. Format = U2.8 (The default is 0)
	1:0	Reserved : MBZ
51	31	Reserved : MBZ
	30:24	**UV Threshold:** Low UV threshold. Format = U7 (The default is 3)
	23:19	Reserved : MBZ
	18:16	**UV Threshold Bits:** Low UV transition width bits. Format = U3 (The default is 3)
	15:13	Reserved : MBZ
	12:8	**STE Threshold:** Skin tone pixels enhancement threshold. Format = U5 (The default is 0)
	7:3	Reserved : MBZ
	2:0	**STE Slope Bits:** Skin tone pixels enhancement slope bits. Format = U3 (The default is 0)

DWord	Bit	Description
52	31:16	**Inv_UVMaxColor:** 1 / UVMaxColor. Used for the SFs2 calculation. Format = U0.16 (The default is 146)
	15:9	Reserved : MBZ
	8:0	**UVMaxColor:** The maximum absolute value of the legal UV pixels. Used for the SFs2 calculation. Format = U9 (The default is 448)
		PROCAMP State
53	31:28	Reserved : MBZ
	27:17	**Contrast:** Contrast magnitude. Format = U4.7 (The default is 1.0)
	16:13	Reserved : MBZ
	12:1	**Brightness:** Brightness magnitude. Format = S7.4 2's complement (The default is 0.0)
	0	**PROCAMP Enable** Format = Enable (The default is 1)
54	31:16	**Cos_c_s:** UV multiplication cosine factor. Format = S7.8 2's complement (The default is 256)
	15:0	**Sin_c_s:** UV multiplication sine factor. Format = S7.8 2's complement (The default is 0)
		CSC State
55	31:29	Reserved : MBZ
	28:16	**C1:** Transform coefficient. Format = S2.10 2's complement (The default is 0)
	15:3	**C0:** Transform coefficient. Format = S2.10 2's complement (The default is 1024)
	2	**YUV_IN:** CSC input offset enable Format = YUV (The default is 0)
	1	**YUV_OUT:** CSC output offset enable Format = RGB (The default is 0)
	0	**Transform Enable** Format = Enable
56	31:26	Reserved : MBZ
	25:13	**C3:** Transform coefficient. Format = S2.10 2's complement (The default is 0)
	12:0	**C2:** Transform coefficient. Format = S2.10 2's complement (The default is 0)

DWord	Bit	Description
57	31:26	Reserved : MBZ
	25:13	**C5:** Transform coefficient. Format = S2.10 2's complement (The default is 0)
	12:0	**C4:** Transform coefficient. Format = S2.10 2's complement (The default is 1024)
58	31:26	Reserved : MBZ
	25:13	**C7:** Transform coefficient. Format = S2.10 2's complement (The default is 0)
	12:0	**C6:** Transform coefficient. Format = S2.10 2's complement (The default is 0)
59	31:13	Reserved : MBZ
	12:0	**C8:** Transform coefficient. Format = S2.10 2's complement (The default is 1024)
60	31:20	Reserved : MBZ
	19:10	**Offset out 1:** Offset out for Y/R. Format = S9 2's complement (The default is 0)
	9:0	**Offset in 1:** Offset in for Y/R. Format = S9 2's complement (The default is 0)
61	31:20	Reserved : MBZ
	19:10	**Offset out 2:** Offset out for U/G. Format = S9 2's complement (The default is 0)
	9:0	**Offset in 2:** Offset in for U/G. Format = S9 2's complement (The default is 0)
62	31:20	Reserved : MBZ
	19:10	**Offset out 3:** Offset out for V/B. Format = S9 2's complement (The default is 0)
	9:0	**Offset in 3:** Offset in for V/B. Format = S9 2's complement (The default is 0)
63	31:17	Reserved : MBZ
	16	**Alpha from State Select** Format = U1 enumerated type 0: alpha is taken from message 1: alpha is taken from state
	15:0	**Color Pipe Alpha** Format = U16

3.9 Messages

3.9.1 Global Definitions

For data port messages, part of the message descriptor is used to determine the message type. This field is documented here. The remainder of the message descriptor is defined differently depending on the message type, and is documented in the section for the corresponding message.

[DevSNB+]: The Data Port is actually three separate targets, **Data Port Sampler Cache, Data Port Constant Cache,** and **Data Port Render Cache**, each with its own target unit ID. Each target has its own set of message type encodings as shown below.

Restrictions:

- Data port messages may not have the **End of Thread** bit set in the message descriptor other than the following exeptions:
    - The Render Target Write message may have **End of Thread** set for pixel shader threads dispatched by the windower in non-contiguous dispatch mode.
    - The Render Target UNORM Write message may have **End of Thread** set for pixel shader threads dispatched by the windower in contiguous dispatch mode.
    - **[DevSNB+] only:** The Media Block Write message may have **End of Thread** set for pixel shader threads dispatched by the windower in contiguous dispatch mode.

3.9.2 Data Port Messages

Most of the messages have an existing definition that is not expected to change. There are several new messages that are documented here.

Data Cache Data Port Message Summary

Message Type	Header Required	Shared Local Memory Support	Stateless Support	Address Modes	Vector Width
OWord Block Read	yes	no	yes	global	1
OWord Block Write	yes	no	yes	global	1
Unaligned OWord Block Read	yes	no	yes	global	1
OWord Dual Block Read	no	no	yes	global + offset	2
OWord Dual Block Write	no	no	yes	global + offset	2

Message Type	Header Required	Shared Local Memory Support	Stateless Support	Address Modes	Vector Width
DWord Scattered Read	no	no	yes	global + offset	8, 16
DWord Scattered Write	no	no	yes	global + offset	8, 16
Byte Scattered Read	no	yes		global + offset	8, 16
Byte Scattered Write	no	yes		global + offset	8, 16
Untyped Surface Read	no	yes		1D or 2D	2, 8, 16
Untyped Surface Write	no	yes		1D or 2D	2, 8, 16
Untyped Atomic Operation	no	yes		1D or 2D	8, 16
Scratch Block Read	yes	no	yes (only)	Imm_Buf + offset	
Scratch Block Write	yes	no	yes (only)	Imm_Buf + offset	
Memory Fence	yes	N/A	N/A	N/A	N/A

"global" is the **Global Offset** in the message header (if header is not present, Global Offset is zero).
"imm_buf" is the Immediate Buffer Base Address provided in message header register M0.5.
"offset" is in the message payload, and is per-slot.
"handle" is the handle address in the message header.
"URBoffset" is the **Global Offset** field in the URB message descriptor.
"1D" and "2D" are the address payload.

[DevSNB+] Render Cache Data Port Message Summary

Message Type	Header Required	Address Modes	Vector Width
Media Block Read	yes	2D	1
Media Block Write	yes	2D	1
Render Target Write	No[1]	2D + RTAI	8, 16

Typed Surface Read	yes	1D, 2D, 3D, 4D	8
Typed Surface Write	yes	1D, 2D, 3D, 4D	8
Typed Atomic Operation	yes	1D, 2D, 3D, 4D	8
Memory Fence	yes	N/A	N/A

"4D" address refers to U/V/R/LOD for mip-mapped surfaces
"2D + RTAI" address refers to a basic 2D address with render target array index for the third dimension

[1][DevSNB-A/B] Errata: Render Target Write messages require a header when Pixel Shader Computed Depth is enabled

3.9.2.1 Message Descriptor

3.9.2.1.1 Message Descriptor [DevSNB+]

The following message descriptor applies to [DevSNB+].

DATA PORT SAMPLER CACHE		DATA PORT CONSTANT CACHE		DATA PORT RENDER CACHE	
Bit	**Description**	**Bit**	**Description**	**Bit**	**Description**
19	**Header Present.** If set, indicates that the message includes the header. Refer to Render Target Write message section for more details on this field. **Programming Notes:** The header must be present unless the message type is *Render Target Write* **Erratum: [DevSNB+]:**SW must not rely on HW to perform out of bounds check for (X,Y) for Render Target Write messages with this bit reset. Format = Enable				
18	Ignored				
17:16	Ignored	17:16	Ignored	17	**Send Write Commit Message.** Indicates that a write commit message will be sent back to the thread when the write has been committed. See section 3.3 for more details. This field is ignored on read message

DATA PORT SAMPLER CACHE		DATA PORT CONSTANT CACHE		DATA PORT RENDER CACHE	
					types. Format = Enable
15:13	**Message Type** 000: OWord Block Read 010: OWord Dual Block Read 100: Media Block Read 101: Unaligned OWord Block Read 110: DWord Scattered Read All other encodings are reserved.	15:13	**Message Type** 000: OWord Block Read 010: OWord Dual Block Read 110: DWord Scattered Read All other encodings are reserved.	16:13	**Message Type** 0000: OWord Block Read 0001: Render Target UNORM Read 0010: OWord Dual Block Read 0100: Media Block Read 0101: Unaligned OWord Block Read 0110: DWord Scattered Read 0111: DWord Atomic write message 1000: OWord Block Write 1001: OWord Dual Block Write 1010: Media Block Write 1011: DWord Scattered Write 1100: Render Target Write 1101: Streamed Vertex Buffer Write 1110: Render Target UNORM Write All other encodings are reserved.
12:8	**Message Specific Control.** Refer to the specific message section for the definition of these bits.				
7:0	**Binding Table Index.** Specifies the index into the binding table for the specified surface. A binding table index of 255 indicates that a stateless model is to be used. The stateless model is allowed only with the render cache data port. Refer to section 2.2.2 for details on the stateless model. Format = U8				

DATA PORT SAMPLER CACHE	DATA PORT CONSTANT CACHE	DATA PORT RENDER CACHE
Range = [0,255]		

3.9.2.2 Message Header

This header applies to the following data port messages:

- OWord Block Read/Write
- Unaligned OWord Block Read
- OWord Dual Block Read/Write
- DWord Scattered Read/Write

The header definitions for the other data port messages is in the section for each message.

DWord	Bit	Description
M0.7	31:0	**Reserved**
M0.6	31:0	**Reserved**
M0.5	31:10	**Immediate Buffer Base Address.** Specifies the surface base address for messages in which the Binding Table Index is 255 (stateless model), otherwise this field is ignored. This pointer is relative to the **General State Base Address**. Format = GeneralStateOffset[31:10]
	9:8	Ignored
	7:0	**Dispatch ID.** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion.
M0.4	31:0	Ignored (reserved for hardware delivery of binding table pointer)
M0.3	31:4	Ignored
	3:0	**Reserved**

DWord	Bit	Description
M0.2	31:0	**Global Offset.** **[DevSNB+]:** Specifies the global element offset into the buffer. • For the Unaligned OWord messages, this offset is in units of Bytes but must be DWord aligned (bits 1:0 MBZ) • For the other OWord messages, this offset is in units of OWords • For the DWord messages, this offset is in units of DWords • For the Byte messages, this offset is in units of Bytes Format = U32 Range = [0,FFFFFFFCh] for Unaliged OWord messages Range = [0,0FFFFFFFh] for other OWord messages Range = [0,3FFFFFFFh] for DWord messages • Range = [0,FFFFFFFFh] for Byte messages Format = U32 Range = [0,FFFFFFFCh] for Unaliged OWord messages Range = [0,0FFFFFFFh] for other OWord messages Range = [0,3FFFFFFFh] for DWord messages
M0.1	31:0	Ignored
M0.0	31:0	Ignored

3.9.2.3 Write Commit Writeback Message

The writeback message is only sent on Data Port Write messages if the **Send Write Commit Message** bit in the message descriptor is set. The destination register is not modified. Write messages without the **Send Write Commit Message** bit set will not return anything to the thread (response length is 0 and destination register is null).

DWord	Bit	Description
W0.7:0		Reserved

3.9.3 OWord Block Read/Write

This message takes one offset (Global Offset), and reads or writes 1, 2, 4, or 8 contiguous OWords starting at that offset.

Restrictions:

- the only surface type allowed is SURFTYPE_BUFFER.
- the surface format is ignored, data is returned from the constant buffer to the GRF without format conversion.

- the surface pitch is ignored, the surface is treated as a 1-dimensional surface. An element size (pitch) of 16 bytes is used to determine the size of the buffer for out-of-bounds checking if using the surface state model. [DevSNB+]

- the surface cannot be tiled

- the surface base address must be OWord aligned

- the **Render Cache Read Write Mode** field in SURFACE_STATE must be set to read/write mode when using this message with the render cache in the surface state model

- the **Stateless Render Cache Read-Write Mode** field in the SVG_WORK_CTL register must be set to read/write mode when using this message with the render cache in the stateless model

Applications:

- constant buffer reads of a single constant or multiple contiguous constants

- scratch space reads/writes where the index for each pixel/vertex is the same

- block constant reads, scratch memory reads/writes for media

Execution Mask. The low 8 bits of the execution mask are used to enable the 8 channels in the first and third GRF registers returned (W0, W2) for read, or the first and third write registers sent (M1, M3). The high 8 bits are used similarly for the second and fourth (W1, W3 or M2, M4). For reads, any mask bit asserted within a group of four will cause the entire OWord to be read and returned to the destination GRF register. For writes, each mask bit is considered for its corresponding DWord written to the destination surface.

For the 1-OWord messages, only the low 8 bits of the execution mask are used. Either the low 4 bits or the high 4 bits, depending on the position of the OWord to be read or written, is used as the single group of four with behavior following that in the preceding paragraph. **[DevBW,DevCL] errata:** Execution mask bits outside of those corresponding to the OWord being read/written cannot be asserted.

The above behavior enables a SIMD16 thread to use the 8-OWord form of this message to access two channels (red and green) of a single scratch register across 16 pixels. A second message would access the other two channels (blue and alpha). The execution mask is used to ensure that data associated with inactive pixels are not overwritten.

Out-of-Bounds Accesses. Reads to areas outside of the surface return 0. Writes to areas outside of the surface are dropped and will not modify memory contents.

3.9.3.1 OWord Block Message Descriptor

Bit	Description
13	**Reserved**
12	Ignored **[DevSNB]**
11	Ignored **[DevSNB]**
10:8	**Block Size.** Specifies the number of contiguous OWords to be read or written

Bit	Description
13	**Reserved**
	000: 1 OWord, read into or written from the low 128 bits of the destination register 001: 1 OWord, read into or written from the high 128 bits of the destination register 010: 2 OWords 011: 4 OWords 100: 8 OWords all other encodings are reserved. **Programming Notes:** • The 6 OWord block size is valid only with **Data Port Constant Cache**.

3.9.3.2 Message Payload (Write)

For the write operation, the message payload consists of one, two, or four registers (not including the header) depending on the **Block Size** specified in the message. For the one-constant case, data is taken from either the high or low half of the payload register depending on the half selected in **Block Size**. In this case, the other half of the payload register is ignored.

The **Offset** referred to below is the **Global Offset** and is in units of OWords. The **OWord** array index is also in units of OWords.

DWord	Bit	Description
M1.7:4	127:0	**OWord[Offset + 1].** If the block size is 1 OWord to be written from the high 128 bits of the destination, OWord[Offset] will appear in this location
M1.3:0	127:0	**OWord[Offset]**
M2.7:4	127:0	**OWord[Offset+3]**
M2.3:0	127:0	**OWord[Offset+2]**
M3.7:4	127:0	**OWord[Offset+5]**
M3.3:0	127:0	**OWord[Offset+4]**
M4.7:4	127:0	**OWord[Offset+7]**
M4.3:0	127:0	**OWord[Offset+6]**

3.9.3.3 Writeback Message (Read)

For the read operation, the writeback message consists of one, two, three, or four registers depending on the **Block Size** specified in the message. For the one-constant case, data is placed in either the high or low half of the returned register depending on the half selected in **Block Size**. In this case, the other half of the register is not changed.

The **Offset** referred to below is the **Global Offset** and is in units of OWords. The **OWord** array index is also in units of OWords.

DWord	Bit	Description
W0.7:4	127:0	**OWord[Offset + 1].** If the block size is 1 OWord to be loaded into the high 128 bits of the destination, OWord[Offset] will appear in this location
W0.3:0	127:0	**OWord[Offset]**
W1.7:4	127:0	**OWord[Offset+3]**
W1.3:0	127:0	**OWord[Offset+2]**
W2.7:4	127:0	**OWord[Offset+5]**
W2.3:0	127:0	**OWord[Offset+4]**
W3.7:4	127:0	**OWord[Offset+7]**
W3.3:0	127:0	**OWord[Offset+6]**

3.9.4 Unaligned OWord Block Read [DevSNB+]

This message takes one DWord aligned offset (**Global Offset**), and reads 1, 2, 4, or 8 contiguous OWords starting at that offset. This message is identical to the OWord Block Read message except the offset alignment. For read/write cache, only the read path supports this unaligned OWord Block access.

Restrictions:

- the only surface type allowed is SURFTYPE_BUFFER.
- the surface format is ignored, data is returned from the constant buffer to the GRF without format conversion.
- the surface pitch is ignored, the surface is treated as a 1-dimensional surface. An element size (pitch) of 16 bytes is used to determine the size of the buffer for out-of-bounds checking if using the surface state model. [DevSNB+]
- the surface cannot be tiled
- the surface base address must be **OWord** aligned
- the **Render Cache Read Write Mode** field in SURFACE_STATE must be set to read/write mode when using this message with the render cache in the surface state model

- the **Stateless Render Cache Read-Write Mode** field in the SVG_WORK_CTL register must be set to read/write mode when using this message with the render cache in the stateless model

Applications:

- Reads with offset that is not aligned with data size, such as row store usage in media

Execution Mask. The execution mask is ignored by this message.

Out-of-Bounds Accesses. Reads to areas outside of the surface return 0.

3.9.4.1 Message Descriptor

Bit	Description
13	Ignored
12:11	Ignored
10:8	**Block Size.** Specifies the number of contiguous OWords to be read 000: 1 OWord, read into the low 128 bits of the destination register 001: 1 OWord, read into the high 128 bits of the destination register 010: 2 OWords 011: 4 OWords 100: 8 OWords all other encodings are reserved.

3.9.4.2 Writeback Message (Read)

For the read operation, the writeback message consists of one, two, or four registers depending on the **Block Size** specified in the message. For the one-constant case, data is placed in either the high or low half of the returned register depending on the half selected in **Block Size**. In this case, the other half of the register is not changed.

The **Global Offset** is in units of **Bytes**, aligned to **DWord** (two LSBs set to zero). The **OWordX** array in units of OWord starts at Global Offset.

DWord	Bit	Description
W0.7:4	127:0	**OWord1 = *(&OWord0 + 1).** If the block size is 1 OWord to be loaded into the high 128 bits of the destination, OWord0 will appear in this location
W0.3:0	127:0	**OWord0 = Buffer[Global Offset]**
W1.7:4	127:0	**OWord3 = *(&OWord2 + 1)**
W1.3:0	127:0	**OWord2 = *(&OWord1 + 1)**
W2.7:4	127:0	**OWord5= *(&OWord4 + 1)**
W2.3:0	127:0	**OWord4 = *(&OWord3 + 1)**
W3.7:4	127:0	**OWord7 = *(&OWord6 + 1)**

DWord	Bit	Description
W3.3:0	127:0	**OWord6 = *(&OWord5 + 1)**

3.9.5 OWord Dual Block Read/Write

This message takes two offsets, and reads or writes 1 or 4 contiguous OWords starting at each offset. The Global Offset is added to each of the specific offsets.

Programming Restrictions: Writes to overlapping addresses will have undefined write ordering.

Restrictions:

- the only surface type allowed is SURFTYPE_BUFFER.
- the surface format is ignored, data is returned from the constant buffer to the GRF without format conversion.
- the surface pitch is ignored, the surface is treated as a 1-dimensional surface. An element size (pitch) of 16 bytes is used to determine the size of the buffer for out-of-bounds checking if using the surface state model. [DevSNB+]
- the surface cannot be tiled
- the surface base address must be OWord aligned
- the **Render Cache Read Write Mode** field in SURFACE_STATE must be set to read/write mode when using this message with the render cache in the surface state model
- the **Stateless Render Cache Read-Write Mode** field in the SVG_WORK_CTL register must be set to read/write mode when using this message with the render cache in the stateless model

Applications:

- SIMD4x2 constant buffer reads where the indices of each vertex/pixel are different (if there are two indices and they are the same, hardware will optimize the cache accesses and do only one cache access)
- SIMD4x2 scratch space reads/writes where the indices are different

Execution Mask. The low 8 bits of the execution mask are used to enable the 8 channels in the GRF registers returned for read, or each of the write registers sent. For reads, any mask bit asserted within a group of four will cause the entire OWord to be read and returned to the destination GRF register. For writes, each mask bit is considered for its corresponding DWord written to the destination surface.

Out-of-Bounds Accesses. Reads to areas outside of the surface return 0. Writes to areas outside of the surface are dropped and will not modify memory contents.

3.9.5.1 Message Descriptor

Bit	Description
13	**Reserved**
12	Ignored
11:10	Ignored
9:8	**Block Size:** Specifies the number of OWords in each block to be read or written 00: 1 OWord 10: 4 OWords all other encodings are reserved.

3.9.5.2 Message Payload

DWord	Bit	Description
M1.7	31:0	Ignored
M1.6	31:0	Ignored
M1.5	31:0	Ignored
M1.4	31:0	**Block Offset 1.** **[DevSNB+]:** Specifies the OWord offset of OWord Block 1 into the surface. Format = U32 Range = [0,0FFFFFFFh]
M1.3	31:0	Ignored
M1.2	31:0	Ignored
M1.1	31:0	Ignored
M1.0	31:0	**Block Offset 0**

3.9.5.3 Additional Message Payload (Write)

For the write operation, the message payload consists of one or four registers (not including the header or the first part of the payload) depending on the **Block Size** specified in the message.

The **Offset1/0** referred to below is the **Global Offset** added to the corresponding **Block Offset 1/0** and is in units of Owords. The **OWord** array index is also in units of OWords.

DWord	Bit	Description
M2.7:4	127:0	**OWord[Offset1]**
M2.3:0	127:0	**OWord[Offset0]**
M3.7:4	127:0	**OWord[Offset1+1]**
M3.3:0	127:0	**OWord[Offset0+1]**
M4.7:4	127:0	**OWord[Offset1+2]**

DWord	Bit	Description
M4.3:0	127:0	**OWord[Offset0+2]**
M4.7:4	127:0	**OWord[Offset1+3]**
M4.3:0	127:0	**OWord[Offset0+3]**

3.9.5.4 Writeback Message (Read)

For the read operation, the writeback message consists of one or four registers depending on the **Block Size** specified in the message.

The **Offset1/0** referred to below is the **Global Offset** added to the corresponding **Block Offset 1/0** and is in units of Owords. The **OWord** array index is also in units of OWords.

DWord	Bit	Description
W0.7:4	127:0	**OWord[Offset1]**
W0.3:0	127:0	**OWord[Offset0]**
W1.7:4	127:0	**OWord[Offset1+1]**
W1.3:0	127:0	**OWord[Offset0+1]**
W2.7:4	127:0	**OWord[Offset1+2]**
W2.3:0	127:0	**OWord[Offset0+2]**
W3.7:4	127:0	**OWord[Offset1+3]**
W3.3:0	127:0	**OWord[Offset0+3]**

3.9.6 Media Block Read/Write

The read form of this message enables a rectangular block of data samples to be read from the source surface and written into the GRF. The write form enables data from the GRF to be written to a rectangular block.

Restrictions:

- the only surface type allowed is non-arrayed, non-mipmapped SURFTYPE_2D. Because of this, the stateless surface model is not supported with this message.
- the surface format is used to determine the pixel structure for boundary clamp, the raw data from the surface is returned to the thread without any format conversion nor filtering operation
- the target cache cannot be the data cache
- the surface base address must be 32-byte aligned
- When a surface is XMajor tiled, (**tile walk** field in the surface state is set to TILEWALK_XMAJOR), a memory area mapped through the Render Cache cannot be read and/or wrote in mixed frame and field modes. For example, if a memory location is first written

with a zero Vertical Line Stride (frame mode), and later on (without render cache flush) read back using Vertical Line Stride of one (field mode), the read data stored in GRF are uncertain.

- The block width and offset should be aligned to the size of pixels stored in the surface. For a surface with 8bpp pixels for example, the block width and offset can be byte aligned. For a surface with 16bpp pixels, it is word aligned.
    - For YUV422 formats, the block width and offset must be pixel pair aligned (i.e. dword aligned).
- The write form of message has the additional restriction that both **X Offset** and **Block Width** must be DWord aligned.
- **[DevSNB-A] Erratum**: IECP enabled medis write messages are not supported.
- When Color Processing is enabled for media write message. Render target must be tiled.

Applications:

- Block reads/writes for media

Execution Mask. The execution mask on the send instruction for this type of message is ignored. The data that is read or written is determined completely by the block parameters.

Out-of-Bounds Accesses. Reads outside of the surface results in the address being clamped to the nearest edge of the surface and the pixel in the position being returned. Writes outside of the surface are dropped and will not modify memory contents.

Determining the boundary pixel value depends on the surface format. Surface format definitions can be found in the Surface Formats Section of the Sampling Engine Chapter.

- For a surface with 8bpp pixels, the boundary byte is replicated. For example, for a boundary dword B0B1B2B3, to replicate the left boundary byte pixel, the out of bound dwords have the format of B0B0B0B0, and that for right boundary is B3B3B3B3.
    - This rule applies to all surface formats with BPE of 8. As the data port does not perform format conversion, the most likely used surface formats are R8_UINT and R8_SINT.
- For any other surfaces with 16bpp pixels, boundary pixel replication is on words. For example, for a boundary dword B0B1B2B3, to replicate the left boundary word pixel, the out of bound dwords have the format of B0B1B0B1, and that for right boundary is B2B3B2B3.
    - This rule applies to all surface formats with BPE of 16. As the data port does not perform format conversion, only the formats with integer data types may be useful in practice.
- For special surfaces with 16bpp pixels YUV422 packed format, there are two basic cases depending on the Y location: YUYV (surface format YCRCB_NORMAL) and UYVY (surface format YCRCB_SWAPY). Boundary handling for YVYU (surface format YCRCB_SWAPUV) is the same as that for YUYV. Similarly, boundary handling for VYUY (surface format YCRCB_SWAPUVY) is the same as that for UYVY. Note that these four surface formats have 16bpp pixels, even though the BPE fields are set to zero according to the table in the Surface Formats Section.

- For a boundary dword Y0U0Y1V0, to replicate the left boundary, we get Y0U0**Y0**V0, and to replicate the right boundary, we get **Y1**U0Y1V0.
- For a boundary dword U0Y0V0Y1, to replicate the left boundary, we get U0Y0V0**Y0**, and to replicate the right boundary, we get U0**Y1**V0Y1.

- For a surface with 32bpp pixels, the boundary dword pixel is replicated.
  - This rule applies to all surface formats with BPE of 32. As the data port does not perform format conversion, some of the formats may not be useful in practice.

Hardware behavior for any other surface types is undefined.

When Color Processing Enable is set to 1 and the IECP output surface to be written is NV12 format (R16_UNORM surface format 0x10A, should be used if the output surface is NV12 format).

1. NV12 surface state : The width of the surface should be always multiples of 4pixels. For 16bpp input message (422 8-bit) the width will always need to be in multiples of 8bytes and for 32bpp input message (422 16-bit or 444 8-bit) the width should be in multiples of 16bytes. Height should be in multiples of 2pixel high. (presently the MFX restriction is that width should be in multiples of 2pixels).
   a. y-offset of the media block write from the EU should be always even
   b. x-offset of the media block write from the EU should be in multiples of 4 pixel.
2. The media block dword write can have only the following combinations (for IECP when NV12 output format is used):
   a. 8pixel wide for 422 8-bit mode
   b. 4pixel wide for 422 8-bit mode
   c. 4pixel wide for 422 16-bit
   d. 4pixel wide for 444 8-bit.
   e. 444 16-bit input format cannot be supported when the output format is NV12 (s/w should not use this combination).
   f. It has to be in multiples of 2pixel high for all above modes.
3. If 444-format is used then we use only the pixel_0 UV values of the 2x2 pixel and the rest are dropped and in case of 422-format the top UV values are used and the bottom UV values is dropped if the output format is NV12 format.
4. Assuming IECP messages will always have vertical stride = 0. (since this is only for pre-processing before the encoder).

3.9.6.1 Message Descriptor

Bit	Description
13	Reserved: MBZ
12	Reserved : MBZ
11	**[DevSNB+]:** Reserved : MBZ
10	**Vertical Line Stride Override** Specifies whether the **Vertical Line Stride** and **Vertical Line Stride Offset** fields in the surface state should be replaced by bits 9 and 8 below. If this field is 1, **Height** in the surface state (see SURFACE_STATE section of Sampling Engine chapter) is modified according the following rules: **Vertical Line Stride** (in surface state) / **Override Vertical Line Stride** / Derived 1-based surface height (As a function of the 0-based **Height** in surface state) 0 / 0 / **Height** + 1 (Normal) 0 / 1 / (**Height** +1) / 2 *Restriction: (Height + 1) must be an even number.* 1 / 0 / (**Height** + 1) * 2 1 / 1 / **Height** + 1 (Normal) For example, for a 720x480 standard resolution video buffer, if **Vertical Line Stride** in surface state is 0, i.e. a frame, **Height** (of the frame) should be 479. When accessing the bottom field of this frame video buffer, both Override Vertical Line Stride and Override Vertical Line Stride Offset will be set to 1, then the derived surface height (of the field) will be 240 ((Height + 1) / 2). In contrary, if Vertical Line Stride in surface state is 1 and Vertical Line Stride Offset in surface state is 0, the surface state represents the top field of the video buffer. In this case, **Height** (of the top field) should be programmed as 239. Accessing the bottom video field will use the same surface height of 240. Accessing the video frame (with Override Vertical Line Stride and Override Vertical Line Stride Offset set to 0) will result in a derived surface height of 480 ((**Height** + 1) * 2). 0 -- Use parameters in the surface state and ignore bits 9:8 1 -- Use bits 9:8 to provide the **Vertical Line Stride** and **Vertical Line Stride Offset**

Bit	Description
9	**Override Vertical Line Stride** Specifies number of lines (0 or 1) to skip between logically adjacent lines – provides support of interleaved (field) surfaces as textures. Format = U1 in lines to skip between logically adjacent lines **[DevBW-A] Erratum**: This field is ignored by hardware.
8	**Override Vertical Line Stride Offset** Specifies the offset of the initial line from the beginning of the buffer. Ignored when **Override Vertical Line Stride** is 0. Format = U1 in lines of initial offset (when Vertical Line Stride == 1)

3.9.6.2 Message Header

DWord	Bit	Description
M0.7	31:0	**Reserved**
M0.6	31:0	**Reserved**
M0.5	31:8	Ignored
	7:0	**FFTID.** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion.
M0.4	31:0	Ignored (reserved for hardware delivery of binding table pointer)
M0.3	31:5	**Color Processing State Pointer [DevSNB+].** Defines the pointer to COLOR_PROCESSING_STATE. Ignored on read messages and when **Color Processing Enable** is not set. This pointer is relative to the **General State Base Address**. **Programming Notes:** • This pointer is *not* delivered via state variables like most other pointers are delivered. It must be delivered via another software-defined mechanism such as CURBE. Format = GeneralStateOffset[31:5]
	4	**Ignored**
	3:2	**Message Format [DevSNB+].** Defines the format of the message if **Color Processing Enable** is set. 0: YUV 4:2:2, 8 bits per channel 1: YUV 4:4:4, 8 bits per channel 2: YUV 4:2:2, 16 bits per channel 3: YUV 4:4:4, 16 bits per channel
	1	**Area of Interest [DevSNB+].** This field controls whether the statistic for the luma pixels is collected at VSC for ACE histogram. This field is effective only when the state variable **Full_image_histogram** is disabled.
	0	**Color Processing Enable [DevSNB+].** This field controls whether color processing is enabled on a media block write message. Format = Enable **[DevSNB-A] Erratum**: This bit must be set to zero.
The following M0.2 definition applies only if the **Message Mode** field is set to NORMAL:		
M0.2	31:29	Ignored
	28:24	**Reserved**

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td></td><td>21:16</td><td>Block Height. Height in rows of block being accessed.
Programming Notes:
• The Block Height is restricted to the following maximum values depending on the Block Width:
<table>
<tr><th>Block Width (bytes)</th><th>Maximum Block Height (rows)</th></tr>
<tr><td>1-4</td><td>64</td></tr>
<tr><td>5-8</td><td>32</td></tr>
<tr><td>9-16</td><td>16</td></tr>
<tr><td>17-32</td><td>8</td></tr>
</table>
Format = U6
Range = [0,63] representing 1 to 64 rows</td></tr>
<tr><td></td><td>15:10</td><td>Ignored</td></tr>
<tr><td></td><td>9:8</td><td>Reserved</td></tr>
<tr><td></td><td>7:5</td><td>Ignored</td></tr>
<tr><td></td><td>4:0</td><td>Block Width. Width in bytes of the block being accessed.
Programming Notes:
• Must be DWord aligned for the write form of the message.
Format = U5
Range = [0,31] representing 1 to 32 Bytes</td></tr>
<tr><td colspan="3">The following M0.2 definition applies only if the Message Mode field is set to PIXEL_MASK:</td></tr>
<tr><td>MO.2</td><td>31:0</td><td>Pixel Mask. One bit per pixel (each pixel being a DWord) indicating which pixels are to be written. This field is ignored by the read message, all pixels are always returned..
The bits in this mask correspond to the pixels (DWords) as follows:
<table>
<tr><td>0</td><td>1</td><td>4</td><td>5</td><td>16</td><td>17</td><td>20</td><td>21</td></tr>
<tr><td>2</td><td>3</td><td>6</td><td>7</td><td>18</td><td>19</td><td>22</td><td>23</td></tr>
<tr><td>8</td><td>9</td><td>12</td><td>13</td><td>24</td><td>25</td><td>28</td><td>29</td></tr>
<tr><td>10</td><td>11</td><td>14</td><td>15</td><td>26</td><td>27</td><td>30</td><td>31</td></tr>
</table></td></tr>
<tr><td>M0.1</td><td>31:0</td><td>Y offset. The Y offset of the upper left corner of the block into the surface.
Format = S31
Programming Notes:
If Message Mode is set to PIXEL_MASK, this field must be a multiple of 4</td></tr>
</table>

DWord	Bit	Description
M0.0	31:0	**X offset.** The X offset of the upper left corner of the block into the surface. Must be DWord aligned (Bits 1:0 MBZ) for the write form of the message. The **X offset** field defines the offset in the input message block. This may differ from the offset in the surface if Color Processing is enabled due to format conversion. **[DevBW, DevCL]** This field must also be DWord aligned for the read form of the message. Format = S31 **Programming Notes:** If **Message Mode** is set to PIXEL_MASK, this field must be a multiple of 32

Programming Note: The legal combinations of block width, pitch control, sub-register offset and block height are given below:

Block Height for given block width, pitch control, subreg offsets

		sub-register offsets							
block width	pitch control	0	1	2	3	4	5	6	7
1-4	00	1-64	1	1	1	1	1	1	1
	01	1-64	1-64	illegal	illegal	1-2	1-2	illegal	illegal
	10	illegal	illegal	illegal	illegal	illegal	illegal	illegal	illegal
	11	1-64	1-64	1-64	1-64	illegal	illegal	illegal	illegal
5-8	00	1-32	illegal	1	illegal	1	illegal	1	illegal
	01	1-32	illegal	1-32	illegal	illegal	illegal	illegal	illegal
	10	illegal	illegal	illegal	illegal	illegal	illegal	illegal	illegal
	11	1-32	illegal	1-32	illegal	1-32	illegal	1-32	illegal
9-16	00	1-16	illegal	illegal	illegal	1	illegal	illegal	illegal
	01	1-16	illegal	illegal	illegal	1-16	illegal	illegal	illegal
	10	illegal	illegal	illegal	illegal	illegal	illegal	illegal	illegal
	11	1-16	illegal	illegal	illegal	1-16	illegal	illegal	illegal
7-32	00	1-8	illegal	illegal	illegal	illegal	illegal	illegal	illegal
	01	1-8	illegal	illegal	illegal	illegal	illegal	illegal	illegal
	10	illegal	illegal	illegal	illegal	illegal	illegal	illegal	illegal
	11	1-8	illegal	illegal	illegal	illegal	illegal	illegal	illegal

3.9.6.3 Message Payload (Write)

DWord	Bit	Description
M1:n		**Write Data.** The format of the write data depends on the **Block Height** and **Block Width**. The data is aligned to the least significant bits of the first register, and the register pitch is equal to the next power-of-2 that is greater than or equal to the **Block Width**.

If **Color Processing Enable** is enabled, the write data is divided into pixels according to the **Message Format** field. The fields within each pixel are defined below. For the 4:2:2 modes, each pixel position includes channels for two pixels.

Message Format	**31:24**	**23:16**	**15:8**	**7:0**
YUV 4:2:2, 8 bits per channel	Cr (V)	right pixel lum (Y1)	Cb (U)	left pixel lum (Y0)
YUV 4:4:4, 8 bits per channel	alpha (A)	luminance (Y)	Cb (U)	Cr (V)
	63:48	**47:32**	**31:16**	**15:0**
YUV 4:2:2, 16 bits per channel	Cr (V)	right pixel lum (Y1)	Cb (U)	left pixel lum (Y0)
YUV 4:4:4, 16 bits per channel	alpha (A)	Cr (V)	luminance (Y)	Cb (U)

3.9.6.4 Writeback Message (Read)

DWord	Bit	Description
W0:n		**Read Data [DevSNB+]** The format of the read data depends on the **Block Height** and **Block Width**. The data is aligned to the least significant bits of the first register, and the register pitch is equal to the next power-of-2 that is greater than or equal to the **Block Width**.

3.9.7 DWord Scattered Read/Write

This message takes a set of offsets, and reads or writes 8 or 16 scattered DWords starting at each offset. The Global Offset is added to each of the specific offsets.

Programming Restrictions: Writes to overlapping addresses will have undefined write ordering.

For read messages with X/Y offsets that are outside the bounds of the surface, the address is clamped to the nearest edge of the surface. For write messages with X/Y offsets that are outside the bounds of the surface, the behavior is undefined.

[DevSNB] Hardware does not check for or optimize for cases where offsets are equal or contiguous, thus for optimal performance in these cases a different message may provide higher performance.

Restrictions:

- the only surface type allowed is SURFTYPE_BUFFER.
- the surface format is ignored, data is returned from the constant buffer to the GRF without format conversion.
- the surface pitch is ignored, the surface is treated as a 1-dimensional surface. An element size (pitch) of 16 bytes is used to determine the size of the buffer for out-of-bounds checking if using the surface state model. [DevSNB+]
- the surface cannot be tiled
- the surface base address must be DWord aligned
- the **Render Cache Read Write Mode** field in SURFACE_STATE must be set to read/write mode when using this message with the render cache in the surface state model
- the **Stateless Render Cache Read-Write Mode** field in the SVG_WORK_CTL register must be set to read/write mode when using this message with the render cache in the stateless model

Applications:

- SIMD8/16 constant buffer reads where the indices of each pixel are different (read one channel per message)
- SIMD8/16 scratch space reads/writes where the indices are different (read/write one channel per message)
- general purpose DWord scatter/gathering, used by media

Execution Mask. Depending on the block size, either the low 8 bits or all 16 bits of the execution mask are used to determine which DWords are read into the destination GRF register (for read), or which DWords are written to the surface (for write).

Out-of-Bounds Accesses. Reads to areas outside of the surface return 0. Writes to areas outside of the surface are dropped and will not modify memory contents.

3.9.7.1 Message Descriptor

Bit	Description
13	**Reserved**
12	Ignored
11:10	Ignored

Bit	Description
13	**Reserved**
9:8	**Block Size.** Specifies the number of DWords to be read or written 10: 8 DWords 11: 16 DWords All other encodings are reserved.

3.9.7.2 Message Payload

DWord	Bit	Description
M1.7	31:0	**Offset 7.** **[DevSNB+]:** Specifies the DWord offset of DWord 7 into the surface. Format = U32 Range = [0,3FFFFFFFh]
M1.6	31:0	**Offset 6**
M1.5	31:0	**Offset 5**
M1.4	31:0	**Offset 4**
M1.3	31:0	**Offset 3**
M1.2	31:0	**Offset 2**
M1.1	31:0	**Offset 1**
M1.0	31:0	**Offset 0**
M2.7	31:0	**Offset 15.** This message register is included only if the block size is 16 DWords.
M2.6	31:0	**Offset 14**
M2.5	31:0	**Offset 13**
M2.4	31:0	**Offset 12**
M2.3	31:0	**Offset 11**
M2.2	31:0	**Offset 10**
M2.1	31:0	**Offset 9**
M2.0	31:0	**Offset 8**

3.9.7.3 Additional Message Payload (Write)

For the write operation, either one or two additional registers (depending on the block size) of payload contain the data to be written.

The **Offsetn** referred to below is the **Global Offset** added to the corresponding **Offset n** and is in units of Dwords. The **DWord** array index is also in units of DWords.

DWord	Bit	Description
M3.7	31:0	**DWord[Offset7]**
M3.6	31:0	**DWord[Offset6]**
M3.5	31:0	**DWord[Offset5]**
M3.4	31:0	**DWord[Offset4]**
M3.3	31:0	**DWord[Offset3]**
M3.2	31:0	**DWord[Offset2]**
M3.1	31:0	**DWord[Offset1]**
M3.0	31:0	**DWord[Offset0]**
M4.7	31:0	**DWord[Offset15].** This message register is included only if the block size is 16 DWords
M4.6	31:0	**DWord[Offset14]**
M4.5	31:0	**DWord[Offset13]**
M4.4	31:0	**DWord[Offset12]**
M4.3	31:0	**DWord[Offset11]**
M4.2	31:0	**DWord[Offset10]**
M4.1	31:0	**DWord[Offset9]**
M4.0	31:0	**DWord[Offset8]**

3.9.7.4 Writeback Message (Read)

For the read operation, the writeback message consists of either one or two registers depending on the block size.

The **Offsetn** referred to below is the **Global Offset** added to the corresponding **Offset n** and is in units of Dwords. The **DWord** array index is also in units of DWords.

DWord	Bit	Description
W0.7	31:0	**DWord[Offset7]**
W0.6	31:0	**DWord[Offset6]**
W0.5	31:0	**DWord[Offset5]**
W0.4	31:0	**DWord[Offset4]**
W0.3	31:0	**DWord[Offset3]**
W0.2	31:0	**DWord[Offset2]**
W0.1	31:0	**DWord[Offset1]**
W0.0	31:0	**DWord[Offset0]**
W1.7	31:0	**DWord[Offset15].** This writeback message register is included only if the block size is 16 DWords.
W1.6	31:0	**DWord[Offset14]**
W1.5	31:0	**DWord[Offset13]**
W1.4	31:0	**DWord[Offset12]**
W1.3	31:0	**DWord[Offset11]**
W1.2	31:0	**DWord[Offset10]**
W1.1	31:0	**DWord[Offset9]**
W1.0	31:0	**DWord[Offset8]**

3.9.8 DWord Atomic write message [DevSNB]

This message takes a set of offsets, and writes 8 scattered DWords starting at each offset. The Global Offset is added to each of the specific offsets. Although this is a write message, it has the read-data returning based on the atomic opcode.

For offsets that are outside the bounds of the surface, the corresponding DW is turned off in the hardware.

Hardware does not check for or optimize for cases where offsets are equal or contiguous, thus for optimal performance in these cases a different message may provide higher performance.

Restrictions:

- the only surface type allowed is SURFTYPE_BUFFER.

- the surface format is ignored, data is returned to the GRF without format conversion.
- the surface pitch is ignored, the surface is treated as a 1-dimensional surface. An element size (pitch) of 16 bytes is used to determine the size of the buffer for out-of-bounds checking if using the surface state model.
- the surface cannot be tiled
- the surface base address must be DWord aligned

Execution Mask. 8 dword enables are generated out of execution masks.

Out-of-Bounds Accesses. Reads to areas outside of the surface return 0. Writes to areas outside of the surface are dropped and will not modify memory contents.

3.9.8.1 Message Descriptor

Bit	Description
12	**Two-Source Message**. When this bit is set, there are two data-phases for two sources. Two-source message is used only for opcode "0111" and for all other opcodes this bit must be 0. When this bit is 0, M3 is not sent to the data-port.
11:8	**Atomic Operation Code: (Please refer to the table below)** **Unsupported opcodes:** **1101, 1110, 1111**

Opcode	Operation	Return Value
0000	ADD: new = old + src0	Old value
0001	SUB: new = old – src0	Old value
0010	INC : new = old+1	Old value
0011	DEC: new = old-1	Old value
0100	MIN: new = min(old, src0)	Old value
0101	MAX: new = max(old, src0)	Old value
0110	XCHG: new = src0	Old value
0111	CMPXCHG : new = (old==src1) ? src0 : old	Old value
1000	AND: new = old & src0	Old value
1001	OR: new = old | src0	Old value

1010	XOR: new = old ^ src0	Old value
1011	MIN_SINT: new = min(old, src0)	Old value(signed)
1100	MAX_SINT: new = max(old, src0)	Old value(signed)
1101-1111	NOP : new = old,	Old value

3.9.8.2 Message Payload

DWord	Bit	Description
M1.7	31:0	**Offset 7.** Specifies the DWord offset of DWord 7 into the surface. Format = U32 Range = [0,3FFFFFFFh]
M1.6	31:0	**Offset 6**
M1.5	31:0	**Offset 5**
M1.4	31:0	**Offset 4**
M1.3	31:0	**Offset 3**
M1.2	31:0	**Offset 2**
M1.1	31:0	**Offset 1**
M1.0	31:0	**Offset 0**

3.9.8.3 Source Payload

Either one or two additional registers (depending on **Two-Source Message**) of source payload contain the data to be used as source.

The **Offsetn** referred to below is the **Global Offset** added to the corresponding **Offset n** and is in units of DWords. The **DWord** array index is also in units of DWords.

DWord	Bit	Description
M2.7	31:0	**DWord[Offset7] Src0**
M2.6	31:0	**DWord[Offset6] Src0**
M2.5	31:0	**DWord[Offset5] Src0**
M2.4	31:0	**DWord[Offset4] Src0**
M2.3	31:0	**DWord[Offset3] Src0**
M2.2	31:0	**DWord[Offset2] Src0**

DWord	Bit	Description
M2.1	31:0	**DWord[Offset1] Src0**
M2.0	31:0	**DWord[Offset0] Src0**
M3.7	31:0	**DWord[Offset7] Src1**
M3.6	31:0	**DWord[Offset6] Src1**
M3.5	31:0	**DWord[Offset5] Src1**
M3.4	31:0	**DWord[Offset4] Src1**
M3.3	31:0	**DWord[Offset3] Src1**
M3.2	31:0	**DWord[Offset2] Src1**
M3.1	31:0	**DWord[Offset1] Src1**
M3.0	31:0	**DWord[Offset0] Src1**

3.9.8.4 Writeback Message

For the read operation, the writeback message consists of either one or two registers depending on the block size.

The **Offsetn** referred to below is the **Global Offset** added to the corresponding **Offset n** and is in units of DWords. The **DWord** array index is also in units of DWords.

DWord	Bit	Description
W0.7	31:0	**DWord[Offset7]**
W0.6	31:0	**DWord[Offset6]**
W0.5	31:0	**DWord[Offset5]**
W0.4	31:0	**DWord[Offset4]**
W0.3	31:0	**DWord[Offset3]**
W0.2	31:0	**DWord[Offset2]**
W0.1	31:0	**DWord[Offset1]**
W0.0	31:0	**DWord[Offset0]**

3.9.9 Render Target Write

This message takes four subspans of pixels for write to a render target. Depending on parameters contained in the message and state, it may also perform a depth and stencil buffer write and/or a render target read for a color blend operation. Additional operations enabled in the Color Calculator state will also be initiated as a result of issuing this message (depth test, alpha test, logic ops, etc.). This message is intended only for use by pixel shader kernels for writing results to render targets.

Restrictions:

- All surface types are allowed.

- **A**

- For SURFTYPE_BUFFER and SURFTYPE_1D surfaces, only the X coordinate is used to index into the surface. The Y coordinate must be zero.

- For SURFTYPE_1D, 2D, 3D, and CUBE surfaces, a **Render Target Array Index** is included in the input message to provide an additional coordinate. The **Render Target Array Index** must be zero for SURFTYPE_BUFFER.

- The surface format is restricted to the set supported as render target. If source/dest color blend is enabled, the surface format is further restricted to the set supported as alpha blend render target.

- The last message sent to the render target by a thread must have the **End Of Thread** bit set in the message descriptor and the dispatch mask set correctly in the message header to enable correct clearing of the pixel scoreboard.

- The stateless model cannot be used with this message (**Binding Table Index** cannot be 255).

- This message can only be issued from a kernel specified in WM_STATE or 3DSTATE_WM (pixel shader kernel), dispatched in non-contiguous mode. Any other kernel issuing this message will cause undefined behavior.

- **[DevCTG+]:** The dual source message cannot be used if the Render Target Rotation field in SURFACE_STATE is set to anything other than RTROTATE_0DEG.

- This message cannot be used on a surface in field mode (**Vertical Line Stride** = 1)

- If multiple SIMD8 Dual Source messages are delivered by the pixel shader thread, each SIMD8_DUALSRC_LO message must be issued *before* the SIMD8_DUALSRC_HI message with the same **Slot Group Select** setting.

- **[DevSNB-**A]:SIMD8 Dual Source Messages are not supported.

- **[DevSNB-A,B]:** Dual Source Messages to the linear RT or MSRT can result in incorrect PS_DEPTH_COUNT.

- **[DevSNB]:** SIMD8 Image Write: Out of bounds write to SURFTYPE_BUFFER with more than 8K elements is undefined.

Execution Mask. The execution mask for render target messages is ignored. Control of which pixels are active is controlled by the **Pixel/Sample Enables** fields in the message header.

Out-of-Bounds Accesses. Accesses to pixels outside of the surface are dropped and will not modify memory contents. However, if the **Render Target Array Index** is out of bounds, it is set to zero and the surface write is not surpressed.

3.9.9.1 Subspan/Pixel to Slot Mapping

The following table indicates the mapping of subspans, pixels, and samples to slots in the pixel shader dispatch depending on the number of samples and message size. This table applies to all devices, however NumSamples = 4X is supported only on [DevSNB].

Pixels are numbered as follows within a subspan:

0 = upper left

1 = upper right

2 = lower left

3 = lower right

sspi = Starting Sample Pair Index (from the message header)

Message Size	Num Samples	Slot Mapping
SIMD16	1X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[1].Pixel[3:0].Sample[0] Slot[11:8] = Subspan[2].Pixel[3:0].Sample[0] Slot[15:12] = Subspan[3].Pixel[3:0].Sample[0]
	4X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1] Slot[11:8] = Subspan[0].Pixel[3:0].Sample[2] Slot[15:12] = Subspan[0].Pixel[3:0].Sample[3]
SIMD8	1X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[1].Pixel[3:0].Sample[0]
	4X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[2*sspi+0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[2*sspi+1]

Restriction:

- **[DevSNB+]:** When SIMD32 or SIMD16 PS threads send render target writes with multiple SIMD8 and SIMD16 messages, the following must hold:
    - All the slots (as described above) must have a corresponding render target write irrespective of the slot's validity. A slot is considered valid when at least one sample is enabled. For example, a SIMD16 PS thread must send two SIMD8 render target writes to cover all the slots.

- PS thread must send SIMD render target write messages with increasing slot numbers. For example, SIMD16 thread has Slot[15:0] and if two SIMD8 render target writes are used, the first SIMD8 render target write must send Slot[7:0] and the next one must send Slot[15:8].

3.9.9.2 Message Descriptor

3.9.9.2.1 Message Descriptor [DevSNB+]

Bit	Description
13	**[DevSNB+]** Ignored
2	**Last Render Target Select.** This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero.
11	**Slot Group Select.** This field selects whether slots 15:0 or slots 31:16 are used for bypassed data. Bypassed data includes the antialias alpha, multisample coverage mask, and if the header is not present also includes the X/Y addresses and pixel enables. For 8- and 16-pixel dispatches, SLOTGRP_LO must be selected on every message. For 32-pixel dispatches, this field must be set correctly for each message based on which slots are currently being processed. 0: SLOTGRP_LO: choose bypassed data for slots 15:0 1: SLOTGRP_HI: choose bypassed data for slots 31:16 This bit must be zero for SIMD8 Image Write message.

Bit	Description
10:8	**Message Type.** This field specifies the type of render target message. For the SIMD8_DUALSRC_xx messages, the low bit indicates which slots to use for the pixel enables, X/Y addresses, and oMask. **Programming Notes:** • SIMD16_REPDATA (**Message Type** = 111) is only supported when accessing tiled memory. Using this Message Type to access linear (untiled) memory is UNDEFINED. 000: SIMD16: SIMD16 single source message 001: SIMD16_REPDATA: SIMD16 single source message with replicated data 010: SIMD8_DUALSRC_LO: SIMD8 dual source message, use slots 7:0 011: SIMD8_DUALSRC_HI: SIMD8 dual source message, use slots 15:8 100: SIMD8_LO: SIMD8 single source message, use slots 7:0 101: SIMD8_IMAGE_WR: SIMD8 2D/3D Image Write, use slots 7:0 [DevSNB Only] **Note:** the above slots indicated are within the 16 slots selected by **Slot Group Select**. If SLOTGRP_HI is selected, the SIMD8 message types above reference slots 23:16 or 31:24 instead of 7:0 or 15:8, respectively. **[DevSNB]:** When Pixel Shader outputs oDepth and PS invocation mode is PERPIXEL, Message Type for Render Target Write must be SIMD8. **Errata: [DevSNB+]:** When Pixel Shader outputs oMask, this message type is not supported: SIMD8 (including SIMD8_DUALSRC_xx).

3.9.9.3 Message Header

The render target write message has a two-register message header.

3.9.9.3.1 Message Header [DevSNB+]

If the header is not present, behavior is as if the message was sent with most fields set to the same value that was delivered in R0 and R1 on the pixel shader thread dispatch. The following fields, which are not delivered in the pixel shader dispatch, behave as if they are set to zero:

- **Render Target Index**
- **Source0 Alpha Present to Render Target**

DWord	Bit	Description
M0.7	31:0	Reserved
M0.6	31:0	Reserved
M0.5	31:10	Ignored
	9:8	**Color Code:** This ID is assigned by the Windower unit and is used to track synchronizng events. Format: Reserved for HW Implementation Use.
	7:0	**FFTID.** The Fixed Function Thread ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion.
M0.4	31:0	Ignored (reserved for hardware delivery of binding table pointer)
M0.3	31:0	Ignored
M0.2	31:3	Ignored
	2:0	**Render Target Index.** Specifies the render target index that will be used to select blend state from BLEND_STATE. Format = U3
M0.1	31:6	**Color Calculator State Pointer.** Specifies the 64-byte aligned pointer to the color calculator state. This pointer is relative to the **General State Base Address**. Format = GeneralStateOffset[31:6]
	5:0	Ignored
M0.0	31	Ignored
	30:27	**Viewport Index.** Specifies the index of the viewport currently being used. Format = U4 Range = [0,15]

DWord	Bit	Description
	26:16	**Render Target Array Index.** Specifies the array index to be used for the following surface types: SURFTYPE_1D: specifies the array index. Range = [0,511] SURFTYPE_2D: specifies the array index. Range = [0,511] SURFTYPE_3D: specifies the "z" or "r" coordinate. Range = [0,2047] SURFTYPE_CUBE: specifies the face identifier. Range = [0,5] SURFTYPE_BUFFER: must be zero. face — Render Target Array Index **+x** — **0** **-x** — **1** **+y** — **2** **-y** — **3** **+z** — **4** **-z** — **5** Format = U11 The **Render Target Array Index** used by hardware for access to the Render Target is overridden with the **Minimum Array Element** defined in SURFACE_STATE if it is out of the range between **Minimum Array Element** and **Depth**. For cube surfaces, a depth value of 5 is used for this determination.
	15	**Front/Back Facing Polygon.** Determines whether the polygon is front or back facing. Used by the render cache to determine which stencil test state to use. 0: Front Facing 1: Back Facing
	14	Ignored
	13	**Source Depth Present to Render Target.** Indicates that source depth is included in the message.
	12	**oMask to Render Target** This bit indicates that oMask data is present in the message and is to be used to mask off samples.
	11	**Source0 Alpha Present to RenderTarget.** This bit indicates that Source0 Alpha (aka o0.a) data is included in RTWrite message. If present, these alpha values are used as inputs to AlphaTest and AlphaToCoverage functions. This is required to meet the API rules when writing to multiple render targets (MRTs). **Programming Notes:** • This bit should not be set when writing to RT0, though sending and using redundant alpha will provide the correct results (at lower performance). • This bit is not supported on Dual-Source Blend message types, as source0 alpha is already included in those messages. • This bit is not supported on replicated data message types.
	10:9	Ignored

DWord	Bit	Description
	8:6	**Starting Sample Pair Index:** indicates the index of the first sample pair of the dispatch Format = U3 **[DevSNB]:** Range = [0,1]
	5:0	Ignored
M1.7	31:16	**Dispatched Pixel/Sample Enables.** One bit per pixel (or sample within pixel) indicating which pixels/samples were originally enabled when the thread was dispatched. This field is only required for the end-of-thread message and on all dual-source messages. The **Dispatched Pixel/Sample Enables** *must be unmodified* from the ones sent when the pixel shader thread was initiated. If the **Dispatched Pixel/Sample Enables** are modified, behavior is undefined. **Multisample Note:** • When operating in PERSAMPLE mode these bits correspond to samples, not pixels. Each subspan slot (4 bits) corresponds to a specific sample location for the subspan. Note that in NUMSAMPLES_1 mode, a pixel and sample are synonomous. When operating in PERPIXEL mode, this field is ignored, and instead the SampleEnableMask (obtained via bypass) are used to clear the Depth Scoreboard.
	15:0	**Pixel/Sample Enables.** One bit per pixel/sample indicating which pixels/samples are still lit based on kill instruction activity in the pixel shader. This mask is used to control actual writes to the color buffer. **Multisample Note:** • When operating in PERSAMPLE mode these bits correspond to samples, not pixels, as the PS is run per-sample. Each subspan slot (4 bits) corresponds to a specific sample location for the subspan. When operating in PERPIXEL mode, these bits still correspond to pixels, as the PS is run per-pixel. Each pixel's mask bit is replicated according to **Number of Multisamples** and combined with other masks to control writes to the multisample locations.
M1.6	31:0	Ignored
M1.5	31:16	**Y3.** Y coordinate for upper-left pixel of subspan 3 (slot 12) Format = U16
	15:0	**X3.** X coordinate for upper-left pixel of subspan 3 (slot 12) Format = U16
M1.4	31:16	**Y2**
	15:0	**X2**
M1.3	31:16	**Y1**
	15:0	**X1**
M1.2	31:16	**Y0**
	15:0	**X0**
M1.1	31:0	Ignored
M1.0	31:0	Ignored

3.9.9.4 Header for SIMD8_IMAGE_WRITE [DevSNB]

DWord	Bit	Description
M0.7	31:0	Reserved
M0.6	31:0	Reserved
M0.5	31:10	Ignored
	9:8	**Color Code:** This ID is assigned by the Windower unit and is used to track synchronizng events. Format: Reserved for HW Implementation Use.
	7:0	**FFTID.** The Fixed Function Thread ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion.
M0.4	31:0	Ignored (reserved for hardware delivery of binding table pointer)
M0.3	31:0	Ignored
M0.2	31:3	Ignored
	2:0	**Render Target Index.** Specifies the render target index that will be used to select blend state from BLEND_STATE. Format = U3
M0.1	31:6	**Color Calculator State Pointer.** Specifies the 64-byte aligned pointer to the color calculator state. This pointer is relative to the **General State Base Address**. Format = GeneralStateOffset[31:6]
	5:0	Ignored
M0.0	31	Ignored
	30:27	**Viewport Index.** Specifies the index of the viewport currently being used. Format = U4 Range = [0,15] SIMD8_IMAGE_WR message type this field is ignored by hardware.

DWord	Bit	Description
	26:16	**Render Target Array Index.** Specifies the array index to be used for the following surface types: SURFTYPE_1D: specifies the array index. Range = [0,511] SURFTYPE_2D: specifies the array index. Range = [0,511] SURFTYPE_3D: specifies the “z” or “r” coordinate. Range = [0,2047] SURFTYPE_CUBE: specifies the face identifier. Range = [0,5] SURFTYPE_BUFFER: must be zero. **face** — **Render Target Array Index** +x — 0 -x — 1 +y — 2 -y — 3 +z — 4 -z — 5 Format = U11 The **Render Target Array Index** used by hardware for access to the Render Target is overridden with the **Minimum Array Element** defined in SURFACE_STATE if it is out of the range between **Minimum Array Element** and **Depth**. For cube surfaces, a depth value of 5 is used for this determination. For SMD8_IMAGE_WRITE : For SURFTYPE_2D, this field must be 0. For SURFTYPE_3D, this field may not be 0 for "Write-3D-Image" operation.
	15:8	Ignored
	7:0	Pixel Maks for SIMD8 messages. 1: Pixel is enabled 0: Pixel is disabled , in this case the corresponding (x,y) should be ignored by hardware.
M1.7	31:16	**Y7: y-coordinate for pixel 7** Format = U16
	15:0	**X7: x-coordinate for pixel 7** Format = U16
M1.6	31:16	**Y6: y-coordinate for pixel 6** Format = U16
	15:0	**X6: x-coordinate for pixel 6** Format = U16
M1.5	31:16	**Y5: y-coordinate for pixel 5** Format = U16

DWord	Bit	Description
	15:0	**X5: x-coordinate for pixel 5** Format = U16
M1.4	31:16	**Y4: y-coordinate for pixel 4** Format = U16
	15:0	**X4: x-coordinate for pixel 4** Format = U16
M1.3	31:16	**Y3: y-coordinate for pixel 3** Format = U16
	15:0	**X3: x-coordinate for pixel 3** Format = U16
M1.2	31:16	**Y2: y-coordinate for pixel 2** Format = U16
	15:0	**X2: x-coordinate for pixel 2** Format = U16
M1.1	31:16	**Y1: y-coordinate for pixel 1** Format = U16
	15:0	**X1: x-coordinate for pixel 1** Format = U16
M1.0	31:16	**Y0: y-coordinate for pixel 0** Format = U16
	15:0	**X0: x-coordinate for pixel 0** Format = U16

3.9.9.5 Source 0 Alpha Payload [DevSNB+]

The source 0 alpha registers, if included, appear in M2 and M3, immediately following the header (if present).

For the SIMD8 single source message, only slot 7:0 data is sent (M2). The source 0 alpha phases are not supported for dual source messages.

DWord	Bit	Description
M2.7	31:0	**Source 0 Alpha for Slot 7** Format = IEEE_Float This and the next register is only included if **Source 0 Alpha Present** bit is set.
M2.6	31:0	**Source 0 Alpha for Slot 6**

DWord	Bit	Description
M2.5	31:0	**Source 0 Alpha for Slot 5**
M2.4	31:0	**Source 0 Alpha for Slot 4**
M2.3	31:0	**Source 0 Alpha for Slot 3**
M2.2	31:0	**Source 0 Alpha for Slot 2**
M2.1	31:0	**Source 0 Alpha for Slot 1**
M2.0	31:0	**Source 0 Alpha for Slot 0**
M3.7	31:0	**Source 0 Alpha for Slot 15**
M3.6	31:0	**Source 0 Alpha for Slot 14**
M3.5	31:0	**Source 0 Alpha for Slot 13**
M3.4	31:0	**Source 0 Alpha for Slot 12**
M3.3	31:0	**Source 0 Alpha for Slot 11**
M3.2	31:0	**Source 0 Alpha for Slot 10**
M3.1	31:0	**Source 0 Alpha for Slot 9**
M3.0	31:0	**Source 0 Alpha for Slot 8**

3.9.9.6 oMask Payload [DevSNB+]

The oMask payload, if present, follows source 0 alpha. The value of 'p' depends on whether the header and source 0 alpha are present.

Sample "n" for that pixel will be killed (not written to the render target or depth buffer) if bit "n" of the oMask is zero. Bits numbers where "n" is larger than the number of multisamples are ignored.

For the SIMD8 messages, only slots 7:0 data is used, or only slots 15:8 depending on the **Message Type** encoding.

DWord	Bit	Description
Mp.7	31:16	**oMask for Slot 15** Format = 16-bit mask This register is only included if **oMask Present** bit is set.
	15:0	**oMask for Slot 14**
Mp.6	31:16	**oMask for Slot 13**

DWord	Bit	Description
	15:0	**oMask for Slot 12**
Mp.5	31:16	**oMask for Slot 11**
	15:0	**oMask for Slot 10**
Mp.4	31:16	**oMask for Slot 9**
	15:0	**oMask for Slot 8**
Mp.3	31:16	**oMask for Slot 7**
	15:0	**oMask for Slot 6**
Mp.2	31:16	**oMask for Slot 5**
	15:0	**oMask for Slot 4**
Mp.1	31:16	**oMask for Slot 3**
	15:0	**oMask for Slot 2**
Mp.0	31:16	**oMask for Slot 1**
	15:0	**oMask for Slot 0**

3.9.9.7 Color Payload: SIMD16 Single Source

3.9.9.7.1 Color Payload: SIMD16 Single Source [DevSNB+]

This payload is included if the Message Type is SIMD16 single source. The value of 'm' depends on whether the header, source 0 alpha, and oMask are present.

DWord	Bit	Description
Mm.7	31:0	**Slot 7 Red.** Specifies the value of the slot's red component. Format = IEEE Float, S31, or U32 depending on the **Surface Format** of the surface being accessed. SINT formats use S31, UINT formats use U32, and all other formats use Float.
Mm.6	31:0	**Slot 6 Red**
Mm.5	31:0	**Slot 5 Red**
Mm.4	31:0	**Slot 4 Red**
Mm.3	31:0	**Slot 3 Red**

DWord	Bit	Description
Mm.2	31:0	**Slot 2 Red**
Mm.1	31:0	**Slot 1 Red**
Mm.0	31:0	**Slot 0 Red**
M(m+1).7	31:0	**Slot 15 Red**
M(m+1).6	31:0	**Slot 14 Red**
M(m+1).5	31:0	**Slot 13 Red**
M(m+1).4	31:0	**Slot 12 Red**
M(m+1).3	31:0	**Slot 11 Red**
M(m+1).2	31:0	**Slot 10 Red**
M(m+1).1	31:0	**Slot 9 Red**
M(m+1).0	31:0	**Slot 8 Red**
M(m+2)		**Slot[7:0] Green.** See Mm definition for slot locations
M(m+3)		**Slot[15:8] Green.** See M(m+1) definition for slot locations
M(m+4)		**Slot[7:0] Blue.** See Mm definition for slot locations
M(m+5)		**Slot[15:8] Blue.** See M(m+1) definition for slot locations
M(m+6)		**Slot[7:0] Alpha.** See Mm definition for slot locations
M(m+7)		**Slot[15:8] Alpha.** See M(m+1) definition for slot locations

3.9.9.8 Color Payload: SIMD8 Single Source

This payload is included if the Message Type is SIMD8 single source or SIMD8 Image Write. For **[DevSNB+]**, the value of 'm' depends on whether the header, source 0 alpha, and oMask are present.

DWord	Bit	Description
Mm.7	31:0	**Slot 7 Red.** Specifies the value of the slot's red component. Format = IEEE Float, S31, or U32 depending on the **Surface Format** of the surface being accessed. SINT formats use S31, UINT formats use U32, and all other formats use Float.
Mm.6	31:0	**Slot 6 Red**
Mm.5	31:0	**Slot 5 Red**

DWord	Bit	Description
Mm.4	31:0	**Slot 4 Red**
Mm.3	31:0	**Slot 3 Red**
Mm.2	31:0	**Slot 2 Red**
Mm.1	31:0	**Slot 1 Red**
Mm.0	31:0	**Slot 0 Red**
M(m+1)		**Slot[7:0] Green.** See Mm definition for slot locations
M(m+2)		**Slot[7:0] Blue.** See Mm definition for slot locations
M(m+3)		**Slot[7:0] Alpha.** See Mm definition for slot locations

3.9.9.9 Color Payload: SIMD16 Replicated Data

This payload is included if the Message Type specifies single source message with replicated data. One set of R/G/B/A data is included in the message, and this data is replicated to all 16 pixels.

This message is legal with color data only (for **[DevSNB+]**, oMask is also legal with this message). The registers for depth, stencil, and antialias alpha data cannot be included with this message, and the corresponding bits in the message header must indicate that these registers are not present.

For **[DevSNB+]**, the value of 'm' depends on whether the header and oMask are present.

Programming Notes:

- This message is allowed only on tiled surfaces

DWord	Bit	Description
Mm.7:4	31:0	Reserved
Mm.3	31:0	**Alpha.** Specifies the value of all slots' alpha channel. Format = IEEE Float, S31, or U32 depending on the **Surface Format** of the surface being accessed. SINT formats use S31, UINT formats use U32, and all other formats use Float.
Mm.2	31:0	**Blue**
Mm.1	31:0	**Green**
Mm.0	31:0	**Red**

3.9.9.10 Color Payload: SIMD8 Dual Source [DevSNB+]

This payload is included if the **Message Type** specifies dual source message. For **[DevSNB+]**, the value of 'm' depends on whether the header, source 0 alpha, and oMask are present.

The dual source message contains only 2 subspans (8 pixels) due to limitations in message length.

DWord	Bit	Description
Mm.7	31:0	**Slot 7 Source 0 Red.** Specifies the value of the slot's red component. Format = IEEE Float, S31, or U32 depending on the **Surface Format** of the surface being accessed. SINT formats use S31, UINT formats use U32, and all other formats use Float.
Mm.6	31:0	**Slot 6 Source 0 Red**
Mm.5	31:0	**Slot 5 Source 0 Red**
Mm.4	31:0	**Slot 4 Source 0 Red**
Mm.3	31:0	**Slot 3 Source 0 Red**
Mm.2	31:0	**Slot 2 Source 0 Red**
Mm.1	31:0	**Slot 1 Source 0 Red**
Mm.0	31:0	**Slot 0 Source 0 Red**
M(m+1)		**Slot[7:0] Source 0 Green.** See Mm definition for slot locations
M(m+2)		**Slot[7:0] Source 0 Blue.** See Mm definition for slot locations
M(m+3)		**Slot[7:0] Source 0 Alpha.** See Mm definition for slot locations
M(m+4)		**Slot[7:0] Source 1 Red.** See Mm definition for slot locations
M(m+5)		**Slot[7:0] Source 1 Green.** See Mm definition for slot locations
M(m+6)		**Slot[7:0] Source 1 Blue.** See Mm definition for slot locations
M(m+7)		**Slot[7:0] Source 1 Alpha.** See Mm definition for slot locations

3.9.9.11 Depth Payload

The depth registers, if included, appear immediately following the color payload.

For the SIMD8 messages, only slot 7:0 data is sent, or only slot 15:8 depending on the **Message Type** encoding. Any complete message register containing ignored data cannot be delivered.

DWord	Bit	Description
Mn.7	31:0	**Source Depth for Slot 7** Format = IEEE_Float This and the next register is only included if **Source Depth Present** bit is set.
Mn.6	31:0	**Source Depth for Slot 6**
Mn.5	31:0	**Source Depth for Slot 5**
Mn.4	31:0	**Source Depth for Slot 4**
Mn.3	31:0	**Source Depth for Slot 3**
Mn.2	31:0	**Source Depth for Slot 2**
Mn.1	31:0	**Source Depth for Slot 1**
Mn.0	31:0	**Source Depth for Slot 0**
M(n+1).7	31:0	**Source Depth for Slot 15**
M(n+1).6	31:0	**Source Depth for Slot 14**
M(n+1).5	31:0	**Source Depth for Slot 13**
M(n+1).4	31:0	**Source Depth for Slot 12**
M(n+1).3	31:0	**Source Depth for Slot 11**
M(n+1).2	31:0	**Source Depth for Slot 10**
M(n+1).1	31:0	**Source Depth for Slot 9**
M(n+1).0	31:0	**Source Depth for Slot 8**
Mk.7	31:0	**Reserved**
Mk.6	31:0	**Destination Depth for Slot 6**
Mk.5	31:0	**Destination Depth for Slot 5**
Mk.4	31:0	**Destination Depth for Slot 4**
Mk.3	31:0	**Destination Depth for Slot 3**
Mk.2	31:0	**Destination Depth for Slot 2**
Mk.1	31:0	**Destination Depth for Slot 1**

DWord	Bit	Description
Mk.0	31:0	**Destination Depth for Slot 0**
M(k+1).7	31:0	**Destination Depth for Slot 15**
M(k+1).6	31:0	**Destination Depth for Slot 14**
M(k+1).5	31:0	**Destination Depth for Slot 13**
M(k+1).4	31:0	**Destination Depth for Slot 12**
M(k+1).3	31:0	**Destination Depth for Slot 11**
M(k+1).2	31:0	**Destination Depth for Slot 10**
M(k+1).1	31:0	**Destination Depth for Slot 9**
M(k+1).0	31:0	**Destination Depth for Slot 8**

3.9.9.12 Message Sequencing Summary

3.9.9.12.1 Message Sequencing Summary [DevSNB+]

This section summarizes the sequencing that occurs for each legal render target write message. All messages have the M0 and M1 header registers if the header is present. If the header is not present, all registers below are renumbered starting with M0 where M2 appears. All cases not shown in this table are illegal.

Key:

s0, s1 = source 0, source 1

1/0 = slots 15:8

3/2 = slots 7:0

sZ = source depth

oM = oMask

Message Type	oMask Present	Source Depth Present	Alpha Present	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14
000	0	0	0	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A					
000	0	0	1	1/0s0A	3/2s0A	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A			
000	0	1	0	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A	1/0sZ	3/2sZ			

Message Type	oMask Present	Source Depth Present	Alpha Present	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14
000	0	1	1	1/0s0A	3/2s0A	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A	1/0sZ	3/2sZ	
000	1	0	0	oM	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A				
000	1	0	1	1/0soA	3/2s0A	oM	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A		
000	1	1	0	oM	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A	1/0sZ	3/2sZ		
000	1	1	1	1/0s0A	3/2s0A	oM	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A	1/0sZ	3/2sZ
001	0	0	0	RGBA												
001	1	0	0	oM	RGBA											
010	0	0	0	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A					
010	0	1	0	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A	1/0sZ				
010	1	0	0	oM	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A				
010	1	1	0	oM	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A	1/0sZ			
011	0	0	0	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A					
011	0	1	0	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A	3/2sZ				
011	1	0	0	oM	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A				
011	1	1	0	oM	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A	3/2sZ			
100	0	0	0	R	G	B	A									
100	0	0	1	s0A	R	G	B	A								
100	0	1	0	R	G	B	A	sZ								
100	0	1	1	s0A	R	G	B	A	sZ							
100	1	0	0	oM	R	G	B	A								
100	1	0	1	s0A	oM	R	G	B	A							
100	1	1	0	oM	R	G	B	A	sZ							
100	1	1	1	s0A	oM	R	G	B	A	sZ						

3.9.10 Render Target UNORM Read/Write [DevCTG] to [DevSNB]

This message is supported on [DevCTG] to [DevSNB] only.

This message reads from or writes to an 8x4 rectangular block of pixels in the render target.

Restrictions:

- the only **Surface Type** allowed is SURFTYPE_2D. Because of this, the stateless surface model is not supported with this message.

- the **Surface Format** must be R8G8B8A8_UNORM, B8G8R8A8_UNORM, R8G8B8X8_UNORM, or B8G8R8X8_UNORM. This is used to determine the pixel structure for boundary clamp, the raw data from the surface is returned to the thread without any format conversion nor filtering operation

- the **Surface Base Address** must be 32-byte aligned

- When a surface is XMajor tiled, (**Tile Walk** field in the surface state is set to TILEWALK_XMAJOR), a memory area mapped through the Render Cache cannot be read and/or written in mixed frame and field modes. For example, if a memory location is first written with a zero Vertical Line Stride (frame mode), and later on (without render cache flush) read back using Vertical Line Stride of one (field mode), the read data stored in GRF are uncertain.

- Unlike the normal "Render Target Write" message, no operations enabled by COLOR_CALC_STATE are supported (alpha blend, alpha test, depth test, stencil, test, logic ops, etc.). **[Pre-DevSNB]:** Depth buffer operations are still possible if under conditions of "promoted depth" as described in the Windower chapter. Non-promoted and computed depth cases are not supported with this message.

- The **Target Cache** for the read message must be the Render Cache.

- **[Pre-DevSNB]:** If this message is issued from a windower dispatched thread, only one Render Target UNORM Write message is allowed in each 32-pixel dispatch thread, two are required in each 64-pixel dispatch thread. This is because the scoreboard is cleared whenever this message is issued.

Execution Mask. The execution mask on the send instruction for this type of message is ignored. The data that is written is determined by the **Pixel Mask**.

Out-of-Bounds Accesses. Writes outside of the surface result are dropped and do not modify memory contents. Reads outside of the surface return zero.

3.9.10.1 Render Target UNORM Message Descriptor

Bit	Description
12	Ignored
11	Ignored
10	**Vertical Line Stride Override** Specifies whether the **Vertical Line Stride** and **Vertical Line Stride Offset** fields in the surface state should be replaced by bits 9 and 8 below. If this field is 1, **Height** in the surface state (see SURFACE_STATE section of Sampling Engine chapter) is modified according the following rules: **Warning: Vertical Line Stride** **Warning:** (in surface state) / **Warning: Override Vertical Line Stride** / **Warning:** Derived 1-based surface height **Warning:** (As a function of the 0-based **Height** in surface state) **Warning:** 0 / **Warning:** 0 / **Warning:** **Height** + 1 **Warning:** (Normal) **Warning:** 0 / **Warning:** 1 / **Warning:** (**Height** +1) / 2 **Warning:** *Restriction: (Height + 1) must be an even number.* **Warning:** 1 / **Warning:** 0 / **Warning:** (**Height** + 1) * 2 **Warning:** 1 / **Warning:** 1 / **Warning:** **Height** + 1 **Warning:** (Normal) For example, for a 720x480 standard resolution video buffer, if **Vertical Line Stride** in surface state is 0, i.e. a frame, **Height** (of the frame) should be 479. When accessing the bottom field of this frame video buffer, both Override Vertical Line Stride and Override Vertical Line Stride Offset will be set to 1, then the derived surface height (of the field) will be 240 ((Height + 1) / 2). In contrary, if Vertical Line Stride in surface state is 1 and Vertical Line Stride Offset in surface state is 0, the surface state represents the top field of the video buffer. In this case, **Height** (of the top field) should be programmed as 239. Accessing the bottom video field will use the same surface height of 240. Accessing the video frame (with Override Vertical Line Stride and Override Vertical Line Stride Offset set to 0) will result in a derived surface height of 480 ((**Height** + 1) * 2). 0 -- Use parameters in the surface state and ignore bits 9:8 1 -- Use bits 9:8 to provide the **Vertical Line Stride** and **Vertical Line Stride Offset**

Bit	Description
9	**Override Vertical Line Stride** Specifies number of lines (0 or 1) to skip between logically adjacent lines – provides support of interleaved (field) surfaces as textures. Format = U1 in lines to skip between logically adjacent lines
8	**Override Vertical Line Stride Offset** Specifies the offset of the initial line from the beginning of the buffer. Ignored when **Override Vertical Line Stride** is 0. Format = U1 in lines of initial offset (when Vertical Line Stride == 1)

3.9.10.2 Message Header

DWord	Bit	Description
M0.7	31:0	**Reserved**
M0.6	31:0	**Reserved**
M0.5	31:8	Ignored
	7:0	**Dispatch ID.** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion.
M0.4	31:0	Ignored (reserved for hardware delivery of binding table pointer)
M0.3	31:0	Ignored
M0.2	31:0	**Pixel Mask.** One bit per pixel indicating which pixels are lit, possibly impacted by kill instruction activity in the pixel shader. This mask is used to control actual writes to the color buffer. This field is ignored by the read message, all pixels are always returned.
M0.1	31:0	**Y offset.** The Y offset of the upper left corner of the block into the surface. Must be 4-row aligned (Bits 1:0 MBZ). Format = S31
M0.0	31:0	**X offset.** The X offset of the upper left corner of the block into the surface. This is a pixel offset assuming a 32-bit pixel. Must be 8-pixel aligned (Bits 2:0 MBZ). Format = S31

3.9.10.3 Message Payload (Write Message only)

The channels are defined as follows depending on surface format:

Channel	R8G8B8A8_UNORM R8G8B8X8_UNORM	B8G8R8A8_UNORM B8G8R8X8_UNORM
Channel 0	Red	Blue
Channel 1	Green	Green
Channel 2	Blue	Red
Channel 3	Alpha	Alpha

Pixels are numbered as follows:

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

DWord	Bit	Description
M1.7	31:24	**Pixel 15 Channel 1** Format = 8-bit UNORM
	23:16	**Pixel 15 Channel 0**
	15:8	**Pixel 14 Channel 1**
	7:0	**Pixel 14 Channel 0**
M1.6		**Pixel 13 & 12 Channel 1/0**
M1.5		**Pixel 7 & 6 Channel 1/0**
M1.4		**Pixel 5 & 4 Channel 1/0**
M1.3		**Pixel 11 & 10 Channel 1/0**
M1.2		**Pixel 9 & 8 Channel 1/0**
M1.1		**Pixel 3 & 2 Channel 1/0**
M1.0		**Pixel 1 & 0 Channel 1/0**
M2.7		**Pixel 31 & 30 Channel 1/0**
M2.6		**Pixel 29 & 28 Channel 1/0**

DWord	Bit	Description
M2.5		**Pixel 23 & 22 Channel 1/0**
M2.4		**Pixel 21 & 20 Channel 1/0**
M2.3		**Pixel 27 & 26 Channel 1/0**
M2.2		**Pixel 25 & 24 Channel 1/0**
M2.1		**Pixel 19 & 18 Channel 1/0**
M2.0		**Pixel 17 & 16 Channel 1/0**
M3.7:0		**Pixels 15:0 Channel 3/2**
M4.7:0		**Pixels 31:16 Channel 3/2**

3.9.10.4 Writeback Message (Read Message only)

DWord	Bit	Description
W0.7	31:24	**Pixel 15 Channel 1** Format = 8-bit UNORM
	23:16	**Pixel 15 Channel 0**
	15:8	**Pixel 14 Channel 1**
	7:0	**Pixel 14 Channel 0**
W0.6		**Pixel 13 & 12 Channel 1/0**
W0.5		**Pixel 7 & 6 Channel 1/0**
W0.4		**Pixel 5 & 4 Channel 1/0**
W0.3		**Pixel 11 & 10 Channel 1/0**
W0.2		**Pixel 9 & 8 Channel 1/0**
W0.1		**Pixel 3 & 2 Channel 1/0**
W0.0		**Pixel 1 & 0 Channel 1/0**
W1.7		**Pixel 31 & 30 Channel 1/0**
W1.6		**Pixel 29 & 28 Channel 1/0**
W1.5		**Pixel 23 & 22 Channel 1/0**

DWord	Bit	Description
W1.4		**Pixel 21 & 20 Channel 1/0**
W1.3		**Pixel 27 & 26 Channel 1/0**
W1.2		**Pixel 25 & 24 Channel 1/0**
W1.1		**Pixel 19 & 18 Channel 1/0**
W1.0		**Pixel 17 & 16 Channel 1/0**
W2.7:0		**Pixels 15:0 Channel 3/2**
W3.7:0		**Pixels 31:16 Channel 3/2**

3.9.11 Streamed Vertex Buffer Write [DevSNB]

This message writes a single 4-tuple of data to a buffer, at a destination index specified in the message header.

Restrictions:

- surface types allowed are SURFTYPE_BUFFER and SURFTYPE_NULL
- surface formats allowed are indicated in the “Streamed Output Vertex Buffers” column of the Surface Formats table in the Sampling Engine chapter
- the surface cannot be tiled
- use of this message with the **End Of Thread** bit set in the message descriptor is not allowed as the Dispatch ID is not included in the message payload.
- the stateless model cannot be used with this message (**Binding Table Index** cannot be 255).
- Both the surface base address and surface pitch must be DWord aligned.

Execution Mask. The low 4 bits of the execution mask are used to enable the 4 channels of the write to the destination surface.

Out-of-Bounds Accesses. Writes to areas outside of the surface are dropped and will not modify memory contents.

3.9.11.1 Message Descriptor

Bit	Description
12	Ignored
11	Ignored
10	**[DevILK+]:** Ignored
9	**[DevILK+]:** Ignored
8	**[DevILK+]:** Ignored

3.9.11.2 Message Payload

DWord	Bit	Description
M0.7	31:0	**Reserved**
M0.6	31:0	**Reserved**
M0.5	31:0	**Destination Index.** Specifies the index into the destination array where the data will be written Format = U32 Range = $[0,2^{27}-1]$
M0.4	31:0	Ignored (reserved for hardware delivery of binding table pointer)
M0.3	31:0	**A Data.** Data for the A channel of the destination Format = IEEE_Float, U32, or S31 matching the surface format of the target surface (no format conversion is performed by hardware)
M0.2	31:0	**B Data.** Data for the B channel of the destination Format = IEEE_Float, U32, or S31 matching the surface format of the target surface (no format conversion is performed by hardware)
M0.1	31:0	**G Data.** Data for the G channel of the destination Format = IEEE_Float, U32, or S31 matching the surface format of the target surface (no format conversion is performed by hardware)
M0.0	31:0	**R Data.** Data for the R channel of the destination Format = IEEE_Float, U32, or S31 matching the surface format of the target surface (no format conversion is performed by hardware)

3.9.12 AVC Loop Filter Read [DevCTG] to [DevSNB]

This message enables a specially formed AVC Loop Filter control data block to read from the source surface, converted via table-look-up and expanded before being written into the GRF.

Restrictions:

- the only surface type allowed is SURFTYPE_BUFFER.
- the surface base address must be dword aligned

Applications:

- Specifically for AVC Loop Filter

Execution Mask. The execution mask on the send instruction for this type of message is ignored. The data that is read is determined completely by the message parameters.

Out-of-Bounds Accesses. Read outside of the surface returns zero.

The source surface contains an array of AVC-LF data structure, each corresponds to a macroblock. The AVC-LF data structure contains 16 dwords as shown in the following table.

DWord	Bit	Description
0	31:24	Reserved : MBZ
	23	**FilterTopMbEdgeFlag**
	22	**FilterLeftMbEdgeFlag**
	21	**FilterInternal4x4EdgesFlag**
	20	**FilterInternal8x8EdgesFlag**
	19	**FieldModeAboveMbFlag**
	18	**FieldModeLeftMbFlag**
	17	**FieldModeCurrentMbFlag**
	16	**MbaffFrameFlag**
	15:8	**VertOrigin**
	7:0	**HorzOrigin**
1	31:30	**bS_h13**
	29:28	**bS_h12**
	27:26	**bS_h11**
	25:24	**bS_h10**

DWord	Bit	Description
	23:22	**bS_v33**
	21:20	**bS_v23**
	19:18	**bS_v13**
	17:16	**bS_v03**
	15:14	**bS_v32**
	13:12	**bS_v22**
	11:10	**bS_v12**
	9:8	**bS_v02**
	7:6	**bS_v31**
	5:4	**bS_v21**
	3:2	**bS_v11**
	1:0	**bS_v01**
2	31:28	**bS_v30_0**
	17:24	**bS_v20_0**
	23:20	**bS_v10_0**
	19:16	**bS_v00_0**
	15:14	**bS_h33**
	13:12	**bS_h32**
	11:10	**bS_h31**
	9:8	**bS_h30**
	7:6	**bS_h23**
	5:4	**bS_h22**
	3:2	**bS_h21**
	1:0	**bS_h20**

DWord	Bit	Description
3	31:28	**bS_h03_0**
	27:24	**bS_h02_0**
	23:20	**bS_h01_0**
	19:16	**bS_h00_0**
	15:12	**bS_v03**
	11:8	**bS_v02**
	7:4	**bS_v01**
	3:0	**bS_v00**
4	31:24	**bIndexBinternal_Y** Internal index B for Y
	23:16	**bIndexBinternal_Y** Internal index A for Y
	15:12	**bS_h03_1**
	11:8	**bS_h02_1**
	7:4	**bS_h01_1**
	3:0	**bS_h00_1**
5	31:24	**bIndexBleft1_Y**
	23:16	**bIndexAleft1_Y**
	15:8	**bIndexBleft0_Y**
	7:0	**bIndexAleft0_Y**
6	31:24	**bIndexBtop1_Y**
	23:16	**bIndexAtop1_Y**
	15:8	**bIndexBtop0_Y**
	7:0	**bIndexAtop0_Y**
7	31:24	**bIndexBleft0_Cb**
	23:16	**bIndexAleft0_Cb**

DWord	Bit	Description
	15:8	**bIndexBinternal_Cb**
	7:0	**bIndexAinternal_Cb**
8	31:24	**bIndexBtop0_Cb**
	23:16	**bIndexAtop0_Cb**
	15:8	**bIndexBleft1_Cb**
	7:0	**bIndexAleft1_Cb**
9	31:24	**bIndexBinternal_Cr**
	23:16	**bIndexAinternal_Cr**
	15:8	**bIndexBtop1_Cb**
	7:0	**bIndexAtop1_Cb**
10	31:24	**bIndexBleft1_Cr**
	23:16	**bIndexAleft1_Cr**
	15:8	**bIndexBleft0_Cr**
	7:0	**bIndexAleft0_Cr**
11	31:24	**bIndexBtop1_Cr**
	23:16	**bIndexAtop1_Cr**
	15:8	**bIndexBtop0_Cr**
	7:0	**bIndexAtop0_Cr**
12	31:2	Reserved : MBZ

DWord	Bit	Description
	1:0	**DisableDeblockingFilterIdc** This is the slice level signal provided as a hint for kernel performance tuning. It is supplied for cases where some slices in a frame have ILDB and some others don't have. In this case, ILDB kernel will be called for all macroblocks in a frame including the ones in the slice that disables ILDB. Setting this bit correctly will ensure that ILDB is not performed on MBs belonging to the slice which has disable deblocking set to 1. For example, kernel may check bit 0, if it is set to 1, no ILDB is performed on the macroblock. 00 - filterInternalEdgesFlag is set equal to 1 01 – disable all deblocking operation, no deblocking parameter syntax element is read; filterInternalEdgesFlag is set equal to 0 10 - macroblocks in different slices are considered not available; filterInternalEdgesFlag is set equal to 1 11 – Reserved (not defined in AVC)
13	31:0	Reserved : MBZ
14	31:0	Reserved : MBZ
15	31:0	Reserved : MBZ

3.9.12.1 Message Descriptor

Bit	Description
12:11	Ignored (**[DevCTG]:** these bits are part of the **Read Message Type** field)
10:8	Ignored

3.9.12.2 Message Header

DWord	Bit	Description
M0.7	31:0	**Reserved**
M0.6	31:0	**Reserved**
M0.5	31:8	Ignored
	7:0	**Dispatch ID.** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion.
M0.4	31:0	Ignored (reserved for hardware delivery of binding table pointer)
M0.3	31:0	Ignored

DWord	Bit	Description
M0.2	31:0	**Global Offset.** Specifies the global byte offset into the buffer. • This offset must be OWord aligned (bits 3:0 MBZ) Format = U32 Range = [0,FFFFFFF0h]
M0.1	31:0	Ignored
M0.0	31:0	Ignored

3.9.12.3 Writeback Message

The writeback message is formed by the data port using the information from the stored surface and integrated lookup tables defining alpha, beta, tc0, and the edge control map.

Many of the fields are passed directly from the stored surface to the writeback message.

IndexA and IndexB index the following tables to populate the alpha and beta values. These tables are used for Y, Cr, and Cb. IndexTop0 values derive AlphaTop0 and BetaTop0, IndexTop1 values derive AlphaTop1 and BetaTop1, and likewise for the Left values.

Table 3-1.Derivation of offset dependent threshold variables α and β from indexA and indexB

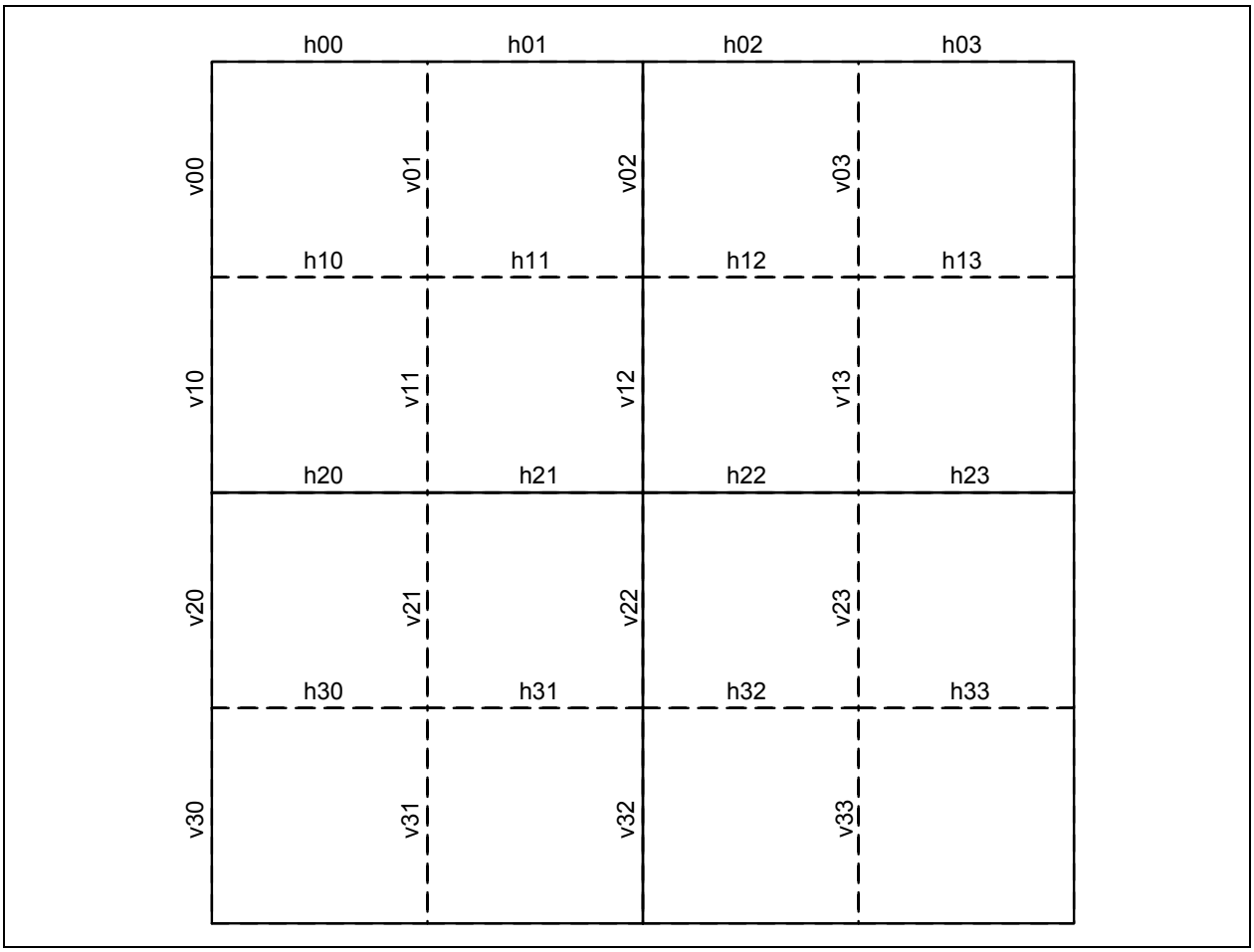
	indexA (for α) or indexB (for β)																								
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
α	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	5	6	7	8	9	10	12	13
β	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	3	3	3	3	4	4	4

Table 3-1. (Concluded) – Derivation of indexA and indexB from offset dependent threshold variables α and β

	indexA (for α) or indexB (for β)																								
	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
α	15	17	20	22	25	28	32	36	40	45	50	56	63	71	80	90	101	113	127	144	162	182	203	226	255	255
β	6	6	7	7	8	8	9	9	10	10	11	11	12	12	13	13	14	14	15	15	16	16	17	17	18	18

For each block boundary, the data port must use the boundary strength values to derive tc0 and an edge control map. The following shows the layout of the boundary values in a Y block. Cr and Cb layout follows suit.

Figure 3-1. Boundary Values Layout in a Y Block



The boundary strengths are used in conjunction with indexA to derive tc0 values. The tables below show tc0 output as a function of the boundary strength (bS) and indexA. On external edges, the boundary strength may be 4. Under this condition, hardware should set the value of tc0 to 0.

For determination of tc0, use IndexA0 and external top and left boundary strength (0) values to derive bTc0 values with an index of _0_. During Mbaff mode, use IndexA1 and external top and left boundary strength (1) to derive bTc0 values with an index of _1_. The layout of the tc0 values in the macroblocks corresponds to Figure 3-1 in the same manner as the boundary strengths.

Table 3-2. Value of variable t_{C0} as a function of indexA and bS

	indexA																									
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
bS = 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
bS = 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
bS = 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
bS = 4	tc0 set to 0																									

Table 3-2 (concluded) – Value of variable t_{C0} as a function of indexA and bS

	indexA																									
	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
bS = 1	1	1	1	1	1	1	1	2	2	2	2	3	3	3	4	4	4	5	6	6	7	8	9	10	11	13
bS = 2	1	1	1	1	1	2	2	2	2	3	3	3	4	4	5	5	6	7	8	8	10	11	12	13	15	17
bS = 3	1	2	2	2	2	3	3	3	4	4	4	5	6	6	7	8	9	10	11	13	14	16	18	20	23	25
bS = 4	tc0 set to 0																									

The boundary strengths also create the edge control maps in the writeback message. The internal boundaries require one control map set according to the boundary strength to drive the deblocking functionality. The external boundaries require two control maps set according to the boundary strength to enable deblocking and choose the deblocking algorithm. These control maps are shown in the tables below. Each edge's boundary strength has a corresponding edge control map (e.g. bS_v01 corresponds to EdgeCntlMap_v01).

Table 3-3. Boundary Strength Mapping to Edge Control Map: Internal Boundaries

bS	Internal boundary Edge Control Map	Description
00	0000	bS = 0, no de-blocking
01	1111	Perform de-blocking using bS < 4 algorithm
10	1111	Perform de-blocking using bS < 4 algorithm
11	1111	Perform de-blocking using bS < 4 algorithm

Table 3-4. Boundary Strength Mapping to Edge Control Map A: External Boundaries, Deblocking Enable

bS	External boundary Edge Control Map A	Description
0000	0000	bS = 0, no de-blocking
0001	1111	bS > 0, de-blocking the segment
0010	1111	bS > 0, de-blocking the segment
0011	1111	bS > 0, de-blocking the segment
0100	1111	bS > 0, de-blocking the segment

Table 3-5. Boundary Strength Mapping to Edge Control Map B: External Boundaries, Deblocking Algorithm

bS	External boundary Edge Control Map B	Description
0000	0000	(No deblocking, set algorithm to 0)
0001	0000	Perform de-blocking using bS < 4 algorithm
0010	0000	Perform de-blocking using bS < 4 algorithm
0011	0000	Perform de-blocking using bS < 4 algorithm
0100	1111	Perform de-blocking using bS = 4 algorithm

The following is the layout of the combined writeback message.

DWord	Bit	Description
W0.7	31:24	**bIndexBleft0_Cb**
	23:16	**bIndexAleft0_Cb**
	15:8	**bIndexBinternal_Cb**

DWord	Bit	Description
	7:0	**bIndexAinternal_Cb**
W0.6	31:24	**bIndexBtop1_Y**
	23:16	**bIndexAtop1_Y**
	15:8	**bIndexBtop0_Y**
	7:0	**bIndexAtop0_Y**
W0.5	31:24	**bIndexBleft1_Y**
	23:16	**bIndexAleft1_Y**
	15:8	**bIndexBleft0_Y**
	7:0	**bIndexAleft0_Y**
W0.4	31:24	**bIndexBinternal_Y** Internal index B for Y
	23:16	**bIndexAinternal_Y** Internal index A for Y
	15:12	**bS_h03_1**
	11:8	**bS_h02_1**
	7:4	**bS_h01_1**
	3:0	**bS_h00_1**
W0.3	31:28	**bS_h03_0**
	27:24	**bS_h02_0**
	23:20	**bS_h01_0**
	19:16	**bS_h00_0**
	15:12	**bS_v30_1**
	11:8	**bS_v20_1**
	7:4	**bS_v10_1**
	3:0	**bS_v00_1**
W0.2	31:28	**bS_v30_0**

DWord	Bit	Description
	27:24	**bS_v20_0**
	23:20	**bS_v10_0**
	19:16	**bS_v00_0**
	15:8	**bbSinternalBotHorz**
	7:0	**bbSinternalMidHorz**
W0.1	31:30	**bS_h13**
	29:28	**bS_h12**
	27:26	**bS_h11**
	25:24	**bS_h10**
	23:22	**bS_v33**
	21:20	**bS_v23**
	19:18	**bS_v13**
	17:16	**bS_v03**
	15:14	**bS_v32**
	13:12	**bS_v22**
	11:10	**bS_v12**
	9:8	**bS_v02**
	7:6	**bS_v31**
	5:4	**bS_v21**
	3:2	**bS_v11**
	1:0	**bS_v01**
W0.0	31:24	Reserved : MBZ
	23	**FilterTopMbEdgeFlag**
	22	**FilterLeftMbEdgeFlag**

DWord	Bit	Description
	21	**FilterInternal4x4EdgesFlag**
	20	**FilterInternal8x8EdgesFlag**
	19	**FieldModeAboveMbFlag**
	18	**FieldModeLeftMbFlag**
	17	**FieldModeCurrentMbFlag**
	16	**MbaffFrameFlag**
	15:8	**VertOrigin**
	7:0	**HorzOrigin**
W1.7	31:0	Reserved : MBZ
W1.6	31:0	Reserved : MBZ
W1.5	31:0	Reserved : MBZ
W1.4	31:0	Reserved : MBZ
W1.3	31:24	**bIndexBtop1_Cr**
	23:16	**bIndexAtop1_Cr**
	15:8	**bIndexBtop0_Cr**
	7:0	**bIndexAtop0_Cr**
W1.2	31:24	**bIndexBleft1_Cr**
	23:16	**bIndexAleft1_Cr**
	15:8	**bIndexBleft0_Cr**
	7:0	**bIndexAleft0_Cr**
W1.1	31:24	**bIndexBinternal_Cr**
	23:16	**bIndexAinternal_Cr**
	15:8	**bIndexBtop1_Cb**
	7:0	**bIndexAtop1_Cb**

DWord	Bit	Description
W1.0	31:24	**bIndexBtop0_Cb**
	23:16	**bIndexAtop0_Cb**
	15:8	**bIndexBleft1_Cb**
	7:0	**bIndexAleft1_Cb**
W2.7	31:28	**EdgeCntlMapB_h03_1** Used in Mbaff mode only
	27:24	**EdgeCntlMapB_h02_1** Used in Mbaff mode only
	23:20	**EdgeCntlMapB_h01_1** Used in Mbaff mode only
	19:16	**EdgeCntlMapB_h00_1** Used in Mbaff mode only
	15:12	**EdgeCntlMapA_h03_1** Used in Mbaff mode only
	11:8	**EdgeCntlMapA_h02_1** Used in Mbaff mode only
	7:4	**EdgeCntlMapA_h01_1** Used in Mbaff mode only
	3:0	**EdgeCntlMapA_h00_1** Used in Mbaff mode only
W2.6	31:28	**EdgeCntlMapB_v30_1** Used in Mbaff mode only
	27:24	**EdgeCntlMapB_v20_1** Used in Mbaff mode only
	23:20	**EdgeCntlMapB_v01_1** Used in Mbaff mode only
	19:16	**EdgeCntlMapB_v00_1** Used in Mbaff mode only
	15:12	**EdgeCntlMapA_v30_1** Used in Mbaff mode only
	11:8	**EdgeCntlMapA_v20_1** Used in Mbaff mode only
	7:4	**EdgeCntlMapA_v10_1** Used in Mbaff mode only

DWord	Bit	Description
	3:0	**EdgeCntlMapA_v00_1** Used in Mbaff mode only
W2.5	31:28	**EdgeCntlMapB_h03_0**
	27:24	**EdgeCntlMapB_h02_0**
	23:20	**EdgeCntlMapB_h01_0**
	19:16	**EdgeCntlMapB_h00_0**
	15:12	**EdgeCntlMapA_h03_0**
	11:8	**EdgeCntlMapA_h02_0**
	7:4	**EdgeCntlMapA_h01_0**
	3:0	**EdgeCntlMapA_h00_0**
W2.4	31:28	**EdgeCntlMapB_v30_0**
	27:24	**EdgeCntlMapB_v20_0**
	23:20	**EdgeCntlMapB_v10_0**
	19:16	**EdgeCntlMapB_v00_0**
	15:12	**EdgeCntlMapA_v30_0**
	11:8	**EdgeCntlMapA_v20_0**
	7:4	**EdgeCntlMapA_v10_0**
	3:0	**EdgeCntlMapA_v00_0**
W2.3	31:0	Reserved : MBZ
W2.2	31:28	**EdgeCntlMap_h33**
	27:24	**EdgeCntlMap_h32**
	23:20	**EdgeCntlMap_h31**
	19:16	**EdgeCntlMap_h30**
	15:12	**EdgeCntlMap_h23**
	11:8	**EdgeCntlMap_h22**

DWord	Bit	Description
	7:4	**EdgeCntlMap_h21**
	3:0	**EdgeCntlMap_h20**
W2.1	31:28	**EdgeCntlMap_h13**
	27:24	**EdgeCntlMap_h12**
	23:20	**EdgeCntlMap_h11**
	19:16	**EdgeCntlMap_h10**
	15:12	**EdgeCntlMap_v33**
	11:8	**EdgeCntlMap_v23**
	7:4	**EdgeCntlMap_v13**
	3:0	**EdgeCntlMap_v03**
W2.0	31:28	**EdgeCntlMap_v32**
	27:24	**EdgeCntlMap_v22**
	23:20	**EdgeCntlMap_v12**
	19:16	**EdgeCntlMap_v02**
	15:12	**EdgeCntlMap_v31**
	11:8	**EdgeCntlMap_v21**
	7:4	**EdgeCntlMap_v11**
	3:0	**EdgeCntlMap_v01**
W3.7	31:24	**bTc0_h33_0_Y**
	23:16	**bTc0_h32_0_Y**
	15:8	**bTc0_h31_0_Y**
	7:0	**bTc0_h30_0_Y**
W3.6	31:24	**bTc0_h23_0_Y**
	23:16	**bTc0_h22_0_Y**

DWord	Bit	Description
	15:8	**bTc0_h21_0_Y**
	7:0	**bTc0_h20_0_Y**
W3.5	31:24	**bTc0_h13_0_Y**
	23:16	**bTc0_h12_0_Y**
	15:8	**bTc0_h11_0_Y**
	7:0	**bTc0_h10_0_Y**
W3.4	31:24	**bTc0_h03_0_Y**
	23:16	**bTc0_h02_0_Y**
	15:8	**bTc0_h01_0_Y**
	7:0	**bTc0_h00_0_Y**
W3.3	31:24	**bTc0_v33_Y**
	23:16	**bTc0_v23_Y**
	15:8	**bTc0_v13_Y**
	7:0	**bTc0_v03_Y**
W3.2	31:24	**bTc0_v32_Y**
	23:16	**bTc0_v22_Y**
	15:8	**bTc0_v12_Y**
	7:0	**bTc0_v02_Y**
W3.1	31:24	**bTc0_v31_Y**
	23:16	**bTc0_v21_Y**
	15:8	**bTc0_v11_Y**
	7:0	**bTc0_v01_Y**
W3.0	31:24	**bTc0_v30_0_Y**
	23:16	**bTc0_v20_0_Y**

DWord	Bit	Description
	15:8	**bTc0_v10_0_Y**
	7:0	**bTc0_v00_0_Y**
W4.7	31:24	**bTc0_h03_1_Y** Used in Mbaff mode only
	23:16	**bTc0_h02_1_Y** Used in Mbaff mode only
	15:8	**bTc0_h01_1_Y** Used in Mbaff mode only
	7:0	**bTc0_h00_1_Y** Used in Mbaff mode only
W4.6	31:24	**bTc0_v30_1_Y** Used in Mbaff mode only
	23:16	**bTc0_v20_1_Y** Used in Mbaff mode only
	15:8	**bTc0_v10_1_Y** Used in Mbaff mode only
	7:0	**bTc0_v00_1_Y** Used in Mbaff mode only
W4.5	31:0	MBZ
W4.4	31:24	**bBetaTop1_Y**
	23:16	**bAlphaTop1_Y**
	15:8	**bBetaLeft1_Y**
	7:0	**bAlphaLeft1_Y**
W4.3	31:0	MBZ
W4.2	31:0	MBZ
W4.1	31:16	MBZ
	15:8	**bBetaInternal_Y**
	7:0	**bAlphaInternal_Y**
W4.0	31:24	**bBetaTop0_Y**

DWord	Bit	Description
W5.7	23:16	**bAlphaTop0_Y**
	15:8	**bBetaLeft0_Y**
	7:0	**bAlphaLeft0_Y**
	31:24	**bTc0_h23_Cr**
	23:16	**bTc0_h22_Cr**
	15:8	**bTc0_h21_Cr**
W5.6	7:0	**bTc0_h20_Cr**
	31:24	**bTc0_h03_0_Cr**
	23:16	**bTc0_h02_0_Cr**
	15:8	**bTc0_h01_0_Cr**
	7:0	**bTc0_h00_0_Cr**
W5.5	31:24	**bTc0_v32_Cr**
	23:16	**bTc0_v22_Cr**
	15:8	**bTc0_v12_Cr**
	7:0	**bTc0_v02_Cr**
W5.4	31:24	**bTc0_v30_0_Cr**
	23:16	**bTc0_v20_0_Cr**
	15:8	**bTc0_v10_0_Cr**
	7:0	**bTc0_v00_0_Cr**
W5.3	31:24	**bTc0_h23_Cb**
	23:16	**bTc0_h22_Cb**
	15:8	**bTc0_h21_Cb**
W5.2	7:0	**bTc0_h20_Cb**
	31:24	**bTc0_h03_0_Cb**

DWord	Bit	Description
W5.1	23:16	**bTc0_h02_0_Cb**
	15:8	**bTc0_h01_0_Cb**
	7:0	**bTc0_h00_0_Cb**
	31:24	**bTc0_v32_Cb**
	23:16	**bTc0_v22_Cb**
	15:8	**bTc0_v12_Cb**
	7:0	**bTc0_v02_Cb**
W5.0	31:24	**bTc0_v30_0_Cb**
	23:16	**bTc0_v20_0_Cb**
	15:8	**bTc0_v10_0_Cb**
W6.7	7:0	**bTc0_v00_0_Cb**
	31:0	MBZ
W6.6	31:0	MBZ
W6.5	31:0	MBZ
W6.4	31:0	MBZ
W6.3	31:16	MBZ
W6.2	15:8	**bBetaInternal_Cr**
	7:0	**bAlphaInternal_Cr**
	31:24	**bBetaTop0_Cr**
	23:16	**bAlphaTop0_Cr**
	15:8	**bBetaLeft0_Cr**
W6.1	7:0	**bAlphaLeft0_Cr**
	31:16	MBZ
	15:8	**bBetaInternal_Cb**

DWord	Bit	Description
W6.0	7:0	**bAlphaInternal_Cb**
	31:24	**bBetaTop0_Cb**
	23:16	**bAlphaTop0_Cb**
	15:8	**bBetaLeft0_Cb**
W7.7	7:0	**bAlphaLeft0_Cb**
	31:24	**bTc0_h03_1_Cr**
	23:16	**bTc0_h02_1_Cr**
	15:8	**bTc0_h01_1_Cr**
W7.6	7:0	**bTc0_h00_1_Cr**
	31:24	**bTc0_v30_1_Cr**
	23:16	**bTc0_v20_1_Cr**
	15:8	**bTc0_v10_1_Cr**
	7:0	**bTc0_v00_1_Cr**
W7.5	31:0	MBZ
W7.4	31:24	**bBetaTop1_Cr**
	23:16	**bAlphaTop1_Cr**
	15:8	**bBetaLeft1_Cr**
W7.3	7:0	**bAlphaLeft1_Cr**
	31:24	**bTc0_h03_1_Cb**
	23:16	**bTc0_h02_1_Cb**
	15:8	**bTc0_h01_1_Cb**
W7.2	7:0	**bTc0_h00_1_Cb**
	31:24	**bTc0_v30_1_Cb**
	23:16	**bTc0_v20_1_Cb**

DWord	Bit	Description
W7.1 W7.0	15:8	**bTc0_v10_1_Cb**
	7:0	**bTc0_v00_1_Cb**
	31:0	MBZ
	31:24	**bBetaTop1_Cb**
	23:16	**bAlphaTop1_Cb**
	15:8	**bBetaLeft1_Cb**
	7:0	**bAlphaLeft1_Cb**

Revision History

Revision	Description	Date
1.0	First 2011 OpenSource edition	May 2011

Intel® OpenSource HD Graphics Programmer's Reference Manual (PRM)

Volume 4 Part 2: Subsystem and Cores – Message Gateway, URB, and IS (SandyBridge)

For the 2011 Intel Core Processor Family

May 2011

Revision 1.0

NOTICE:

This document contains information on products in the design phase of development, and Intel reserves the right to add or remove product features at any time, with or without changes to this open source documentation.

Doc Ref #: IHD-OS-V4 Pt2 – 05 11

Creative Commons License

You are free to Share — to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

No Derivative Works. You may not alter, transform, or build upon this work.

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

Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The SandyBridge chipset family, Havendale/Auburndale chipset family, Intel® 965 Express Chipset Family, Intel® G35 Express Chipset, and Intel® 965GMx Chipset Mobile Family Graphics Controller may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel® sales office or your distributor to obtain the latest specifications and before placing your product order. I2C is a two-wire communications bus/protocol developed by Philips. SMBus is a subset of the I2C bus/protocol and was developed by Intel®. Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

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

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

Copyright © 2011, Intel Corporation. All rights reserved.

Contents

1. Message Gateway 7
1.1 Messages 7
1.1.1 Message Descriptor 8
1.1.2 OpenGateway Message 9
1.1.3 CloseGateway Message 11
1.1.4 ForwardMsg Message 11
1.1.5 GetTimeStamp Message [DevSNB+] 14
1.1.6 BarrierMsg Message 15
2. Unified Return Buffer (URB) 18
2.1 URB Size 18
2.2 URB Access 18
2.3 State 19
2.4 Messages [DevSNB] 19
2.4.1 Execution Mask 20
2.4.2 Message Descriptor 20
2.4.3 URB_WRITE 23
2.4.4 FF_SYNC Messages 28
3. Execution Unit ISA 31
3.1 Introduction 31
3.1.1 Objective and Scope 31
3.1.2 Terms and Acronyms 31
3.1.3 Formats and Conventions 34
4. EU Data Types 35
4.1 Fundamental Data Types 35
4.2 Numerical Data Types 36
4.2.1 Unsigned Integers 38
4.2.2 Signed Integers 38
4.2.3 Single Precision Floating-Point Numbers 38
4.2.4 Packed Signed Half-Byte Integer Vector 38
4.2.5 Packed 8-bit Restricted Float Vector 39
4.3 Floating Point Modes 42
4.3.1 IEEE Floating Point Mode 42
4.3.2 Alternative Floating Point Mode 49
4.4 Type Conversion 50
4.4.1 Float to Integer 50
4.4.2 Integer to Integer with Same or Higher Precision 51
4.4.3 Integer to Integer with Lower Precision 51
4.4.4 Integer to Float 51
5. Execution Environment 52
5.1 Overview 52
5.2 Primary Usage Models 53
5.2.1 AOS and SOA Data Structures 53
5.2.2 SIMD4 Mode of Operation 54
5.2.3 SIMD4x2 Mode of Operation 55
5.2.4 SIMD16 Mode of Operation 57
5.2.5 SIMD8 Mode of Operation 58
5.3 Registers and Register Regions 59
5.3.1 Register Files 59
5.3.2 GRF Registers 59

5.3.3 MRF Registers 61
5.3.4 ARF Registers 61
5.3.5 Immediate 81
5.3.6 Region Parameters 82
5.3.7 Region Addressing Modes 88
5.3.8 Access Modes 93
5.3.9 Execution Data Type 94
5.3.10 Register Region Restrictions 94
5.3.11 Destination Operand Description 97
5.4 SIMD Execution Control 98
5.4.1 Predication 98
5.4.2 No Predication 99
5.4.3 Predication with Horizontal Combination 100
5.4.4 Predication with Vertical Combination 102
5.5 Instruction Compaction 102
5.5.1 Motivation and Expected Usage 102
5.5.2 Hardware Behavior 106
5.5.3 Rules and Restrictions 108
5.6 End of Thread 108
5.7 Creating Conditional Flags 109
5.8 Destination Hazard 111
5.9 Non-present Operands 112
5.10 Instruction Prefetch 112
6. Exceptions 113
6.1 Introduction 113
6.2 Exception-Related Architectural Registers 113
6.3 System Routine 114
6.3.1 General Flow of the System Routine 114
6.3.2 Invoking the System Routine 115
6.3.3 Returning to the Application Thread 115
6.3.4 System-IP (SIP) 116
6.3.5 System Routine Register Space 116
6.3.6 System-Scratch Memory Space 117
6.3.7 Conditional Instructions Within System Routines 117
6.3.8 Messages in System Routines 118
6.3.9 Use of 'NoDDClr' 118
6.4 Exception Descriptions 119
6.4.1 'Illegal' opcode 119
6.4.2 Undefined opcode 119
6.4.3 Software Exception 119
6.4.4 Breakpoint 119
6.4.5 External Halt 119
6.5 Events Which Do Not Generate Exceptions 120
6.6 System Handler Example 121
7. Instruction Set Summary 127
7.1 Instruction Set Characteristics 127
7.1.1 SIMD Instructions and SIMD Width 127
7.1.2 Instruction Operands and Register Regions 127
7.1.3 Instruction Execution 128
7.2 Instruction Machine Formats 128
7.2.1 Common Instruction Fields 132
7.2.2 Instruction Operation Doubleword (DW0) 139
7.2.3 Instruction Destination Doubleword (DW1) 144

7.2.4 Instruction Source-0 Doubleword (DW2) 148
7.2.5 Instruction Source-1 Doubleword (DW3) 153
7.3 Opcode Encoding 156
7.3.1 Move and Logic Instructions 156
7.3.2 Flow Control Instructions 158
7.3.3 Miscellaneous Instructions 158
7.3.4 Parallel Arithmetic Instructions 159
7.3.5 Vector Arithmetic Instructions 160
7.3.6 Special Instructions 161
7.4 Native Instruction BNF 162
7.4.1 Instruction Groups 162
7.4.2 Destination Register 163
7.4.3 Source Register 164
7.4.4 Address Registers 165
7.4.5 Register Files and Register Numbers 166
7.4.6 Relative Location and Stack Control 167
7.4.7 Regions 167
7.4.8 Types 168
7.4.9 Write Mask 168
7.4.10 Swizzle Control 168
7.4.11 Immediate Values 168
7.4.12 Predication and Modifiers 169
7.4.13 Instruction Options 170
7.5 Deprecated Features 171
7.5.1 Defeatured Instructions 171
7.5.2 Others 171
8. Instruction Set Reference 172
8.1 Conventions 172
8.1.1 Pseudo Code Format 172
8.1.2 General Macros and Definitions 172
8.2 Evaluate Write Enable 173
8.3 Instruction Description 174
8.3.1 add – Addition 174
8.3.2 and – Logical And 177
8.3.3 asr – Arithmetic Shift Right 179
8.3.4 avg – Average 180
8.3.5 break – Break [DevSNB] 181
8.3.6 call – Call [DevSNB] 182
8.3.7 case – Case [DevSNB] 184
8.3.8 cmp – Compare 187
8.3.9 cmpn – Compare NaN 188
8.3.10 cont – Continue [DevSNB] 190
8.3.11 dp2 – Dot Product 2 191
8.3.12 dp3 – Dot Product 3 192
8.3.13 dp4 – Dot Product 4 194
8.3.14 dph –Dot Product Homogeneous 195
8.3.15 else – Else [DevSNB] 196
8.3.16 endif – End-If 197
8.3.17 fork – Fork [DevSNB] 199
8.3.18 frc – Fraction 201
8.3.19 halt – Halt [DevSNB] 203
8.3.20 if – If [DevSNB] 204
8.3.21 jmpi – Jump Indexed 206

8.3.22 line – Line 208
8.3.23 lzd – Leading Zero Detection 209
8.3.24 lrp – Linear Interpolation 210
8.3.25 mac – Multiply Accumulate 211
8.3.26 mach – Multiply Accumulate High 212
8.3.27 mad – Multiply Add 214
8.3.28 math – Extended Math Function [DevSNB] 215
8.3.29 mov – Move 218
8.3.30 movi – Move Indexed 219
8.3.31 mul – Multiply 221
8.3.32 nop – No Operation 224
8.3.33 not – Logic Not 225
8.3.34 or – Logic Or 226
8.3.35 pln – Plane 227
8.3.36 ret – Return [DevSNB] 229
8.3.37 rndd – Round Down 230
8.3.38 rndu – Round Up 232
8.3.39 rnde – Round to Even 234
8.3.40 rndz – Round to Zero 236
8.3.41 sad2 – Sum of Absolute Difference 2 238
8.3.42 sada2 – Sum of Absolute Difference Accumulate 2 239
8.3.43 sel – Select 240
8.3.44 send – Send Message [DevSNB] 242
8.3.45 sendc – Conditional Send Message 247
8.3.46 shl – Shift Left 248
8.3.47 shr – Shift Right 249
8.3.48 wait – Wait Notification 250
8.3.49 while – While [DevSNB] 251
8.3.50 xor – Logic Xor 253
9. EU Programming Guide 255
9.1 Assembler Pragmas 255
9.1.1 Declarations 255
9.1.2 Defaults and Defines 256
9.1.3 Example Pragma Usages 257
9.1.4 Assembly Programming Guideline 260
9.2 Usage Examples 262
9.2.1 Vector Immediate 262
9.2.2 Destination Mask for DP4 and Destination Dependency Control 263
9.2.3 Null Register as the Destination 263
9.2.4 Use of LINE Instruction 264
9.2.5 Mask for SEND Instruction 265
9.2.6 Flow Control Instructions 268
9.2.7 Execution Masking 269

1. Message Gateway

The Message Gateway shared function provides a mechanism for active thread-to-thread communication. Such thread-to-thread communication is based on direct register access. One thread, a **requester thread**, is capable of writing into the GRF register space of another thread, a **recipient thread**. Such direct register access between two threads in a multi-processor environment some time is referred to as **remote register access**. Remote register access may include read or write. GEN4 architecture supports **remote register write**, but not remote register read (natively). Message Gateway facilitates such remote register write via message passing. The requester thread sends a message to Message Gateway requesting a write to the recipient thread's GRF register space. Message Gateway sends a writeback message to the recipient thread to complete the register write on behave of the requester. The requester thread and the recipient thread may be on the same EU or on different EUs.

[**DevSNB+**] When Bypass Gateway Control is set to 1, commands **OpenGateway** and **CloseGateway** are no longer used, the gateway parameters are taking the default values as the following:

- **RegBase** = 0
- **Gateway Size** check and **Key** check are bypassed.
- **Gateway Open** (an internal signal that is used to be set by OpenGateway message) check is bypassed

1.1 Messages

Message Gateway supports such thread-to-thread communication with the following three messages:

- **OpenGateway:** opens a gateway for a requester thread. Once a thread successfully opens its gateway, it can be a recipient thread to receive remote register write.

- **CloseGateway:** closes the gateway for a requester thread. Once a thread successfully closes its gateway, Message Gateway will block any future remote register writes to this thread.

- **ForwardMsg:** forwards a formatted message (remote register write) from a requester thread to a recipient thread.

- **GetTimeStamp [DevSNB+]:** reads absolute and relative timestamps for a requester thread.

- **BarrierMsg [DevSNB+]:** a set of threads sends this message to the Gateway. When all threads in a group have sent the message, a reply (both a register write and an N0 notification) is sent to each member of the group.

- **UpdateGatewayState [DevSNB+]:** updates the internal state of the Message Gateway. One example usage is to allow a control thread to change Barrier Byte to convey dynamic state information. This may be used to support interrupt (for advanced scheduler) when persistent compute/worker threads synchronized using Barrier.

1.1.1 Message Descriptor

The following message descriptor applies to all messages supported by Message Gateway.

Bit	Description
19	**Header Present** This bit must be set to **zero** for all Message Gateway messages.
18:17	Ignored
16:15	**Notify.** Send Notification Signal. This is a two-bit field indicating which notify event is sent. 00: No notify 01: Increment recipient thread's N0 notification counter 10: Increment recepient thread's N2 notification counter 11: Reserved This field is only valid for a ForwardMsg message. It is ignored for other messages. The BarrierMsg message always increments the N0 notification counter.
14	**AckReq.** Acknowledgment Required. When this bit is set, an acknowledgment return message is required. Message Gateway will send a writeback message containing the error code to the requester thread using the post destination register address. When this bit is not set, no writeback message is sent to the requesting thread by Message Gateway, even if an error occurs. This field is valid for OpenGateway, CloseGateway, ForwardMsg and BarrierMsg messages. When this bit is set, post destination register must be valid and the response length must be 1. When this bit is not set, post destination register must be *null* and the response length must be 0. This bit cannot be set when EOT is set; otherwise, hardware behavior is undefined. 0 = No Acknowledgement is required. 1 = Acknowledgement is required.
13:3	Reserved: MBZ

Bit	Description
2:0	**SubFuncID.** Identify the supported sub-functions by Message Gateway. Encodings are: 000 = **OpenGateway.** Open the gateway for the requester thread. 001 = **CloseGateway.** Close the gateway for the requester thread. 010 = **ForwardMsg.** Forward the formatted message to the recipient thread with the given offset from the recipient's register base. 011 = **GetTimeStamp [DevSNB+]**. Read absolute and relative timestamps. 100 = **BarrierMsg.** Record an additional thread reaching the barrier. 101 = **UpdateGatewayState.** Update the barrier byte for a barrier. 110 = **MMIO Read/Write** Others are reserved [**Pre-DevSNB**] 011 and 1xx = Reserved

1.1.2 OpenGateway Message

The OpenGateway message opens a communication channel between the requesting thread and other threads. It specifies a key for other threads to access its gateway, as well as the GRF register range allowed to be written. The message consists of a single 256-bit message payload.

If the AckReq bit is set, a single 256-bit payload writeback message is sent back to the requesting thread after completion of the OpenGateway function. Only the least significant DWord in the post destination register is overwritten.

If the EOT is set for this message, Message Gateway will ignore this message; instead, it will close the gateway for the requesting thread regardless of the previous state of the gateway.

It is software's policy to determine how to generate the key.

1.1.2.1 Message Payload

DWord	Bit	Description
M0.7	31:0	**Reserved**
M0.6	31:0	**Reserved**
M0.5	31:29	Reserved: MBZ
	28:21	**RegBase:** The register base address to be stored in the Message Gateway. It is used to compute the destination GRF register address from the offset field in ForwardMsg. RegBase contains 256-bit GRF aligned register address. Note 1: This field aligns with bits [28:21] of the Offset field of the message payload for ForwardMsg. Note 2: the most significant bit of this field must be zero. Format = U8 Range = [0,127]

DWord	Bit	Description
	20:11	Reserved: MBZ
	10:8	[**DevSNB+**] This field is reserved: MBZ
	7:0	**Dispatch ID:** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion. This field is ignored by Message Gateway This field is only required for a thread that is created by a fixed function (therefore, not a child thread) and EOT bit is set for the message.
M0.4	31:16	Reserved: MBZ
	15:0	[**DevSNB+**] This field is reserved: MBZ
M0.3:0		Ignored

1.1.2.2 Writeback Message to Requester Thread

The writeback message is only sent if the **AckReq** bit in the message descriptor is set.

DWord	Bit	Description
W0.7:1		Reserved (not overwritten)
W0.0	31:20	Reserved
	19:16	**Shared Function ID:** Contains the message gateway's shared function ID.
	15:3	Reserved
	2:0	**Error Code** 000: *Successful*. No Error (Normal) 001*: Gateway Size Exceeded*. 101: *Opcode Error.* Attempt to send a message which is not either open/close/forward other codes: Reserved

1.1.3 CloseGateway Message

The CloseGateway message closes a communication channel for the requesting thread that was previously opened with OpenGateway. Each thread is allowed to have only one open gateway at a time, thus no additional information in the message payload is required to close the gateway. The message consists of a single 256-bit message payload.

If the AckReq bit is set, a single 256-bit payload writeback message is sent back to the requesting thread after completion of the CloseGateway function. Only the least significant DWord in the post destination register is overwritten.

1.1.3.1 Message Payload

DWord	Bit	Description
M0.7:6		Ignored
M0.5	31:8	Ignored
	7:0	**Dispatch ID:** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion. This field is ignored by Message Gateway This field is only required for a thread that is created by a fixed function (therefore, not a child thread) and EOT bit is set for the message.
M0.4:0		Ignored

1.1.3.2 Writeback Message to Requester Thread

The writeback message is only sent if the **AckReq** bit in the message descriptor is set.

DWord	Bit	Description
W0.7:1		Reserved (not overwritten)
W0.0	31:20	Reserved
	19:16	**Shared Function ID:** Contains the message gateway's shared function ID.
	15:3	Reserved
	2:0	**Error Code** 000: *Successful*. No Error (Normal) 101: *Opcode Error.* Attempt to send a message which is not either open/close/forward other codes: Reserved

1.1.4 ForwardMsg Message

The ForwardMsg message gives the ability for a requester thread to write a **data segment** in the form of a byte, a dword, 2 dwords, or 4 dwords to a GRF register in a recipient thread. The message consists of a single 256-bit message payload, which contains the specially formatted data segment.

The ForwardMsg message utilizes a communication channel previously opened by the recipient thread. The recipient thread has communicated its EUID, TID, and key to the requester thread previously via

some other mechanism. Generally, this is done through the thread spawn message from parent to child thread, allowing each child (requester) to then communicate with its parent through a gateway opened by the parent (recipient). The child could then use ForwardMsg message to communicate its own EUID, TID, and key back to the parent to enable bi-directional communication after opening its own gateway.

If the AckReq bit is set, a single 256-bit payload writeback message is sent back to the requester thread after completion of the ForwardMsg function. Only the least significant DWord in the post destination register is overwritten.

If the Notify bit in the message descriptor is set, a 'notification' is sent to the recipient thread in order to increment the recipient thread's notification counter. This allows multiple messages to be sent to the recipient without waking up the recipient thread. The last message, having this bit set, will then wake up the recipient thread.

1.1.4.1 Message Payload

DWord	Bit	Description
M0.7	31:0	**Reserved**
M0.6	31:0	**Reserved**
M0.5	31:29	Reserved: MBZ
	28:16	**Offset:** It provides the destination register position in the recipient thread GRF register space as the offset from the RegBase stored in the recipient thread's gateway entry. The offset is in unit of byte, such that bits [28:21] is the 256-bit aligned register offset and bits [4:0] is the sub-register offset. The sub-register offset must be aligned to the Length field in bits [10:8]. The subfields of Offset are further illustrated as the following. Offset[28:21]: Register offset from the gateway base (Range [0, 127]: bit 12 MBZ) Offset[20:18]: DW offset Offset[17:16]: Byte offset (must be 00 for all DW length cases) **Programming restriction:** R0 can not be used as destination GRF register for ForwardMsg. NULL register is also not allowed as destination.
	15:11	Reserved: MBZ
	10:8	**Length:** The length of the data segment. 000: 1 byte 001: 1 word 010: 1 dword 011: 2 dwords 100: 4 dwords 101-111: Reserved
	7:0	**Dispatch ID:** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion. This field is ignored by Message Gateway This field is only required for a thread that is created by a fixed function (therefore, not a child thread) and EOT bit is set for the message.
M0.4	31:30	Ignored
	29	**Reserved**
	28	**Reserved**.

DWord	Bit	Description	
	27:24	**EUID:** The Execution Unit ID as part of the Recipient field is used to identify the recipient thread to whom the message is forwarded.	
	23:19	Ignored	
	18:16	**TID:** The Thread ID as part of the Recipient field is used to identify the recipient thread to whom the message is forwarded.	
	15:0	**Key** The key to match with the one stored in the recipient thread's entry in Message Gateway. [**DevSNB+**] Ignored	
M0.3	31:0	**Data Segment DWord 3:** valid only for the 4-DWord data segment length	
M0.2	31:0	**Data Segment DWord 2:** valid only for the 4-DWord data segment length	
M0.1	31:0	**Data Segment Dword 1:** valid only for the 2- and 4-DWord data segment lengths	
M0.0	31:24	**Data Segment Byte 0:** the same byte must be copied to all four positions within this DWord. Valid only for the 1-Byte data segment length.	**Data Segment Dword 0:** valid only for the 1-, 2- and 4-Dword data segment lengths
	23:16	**Data Segment Byte 0**	
	15:8	**Data Segment Byte 0**	
	7:0	**Data Segment Byte 0**	

1.1.4.2 Writeback Message to Requester Thread

The writeback message is only sent if the **AckReq** bit in the message descriptor is set.

DWord	Bit	Description
W0.7:1		Reserved (not overwritten)
W0.0	31:20	Reserved
	19:16	**Shared Function ID:** Contains the message gateway's shared function ID.
	15:3	Reserved
	2:0	**Error Code** 000: *Successful*. No Error (Normal) 001: Reserved 010: *Gateway Closed*. Attempt to send a message through a closed gateway 011: *Key Mismatched*. [Pre-DevSNB] Attempt to send a message with a mismatching key 100: *Limit Exceeded*. [Pre-DevSNB] Attempt to send a message with offset beyond the gateway limit 101: *Opcode Error.* Attempt to send a message which is not either open/close/forward 110: *Invalid Message Size.* Attempt to forward a message with length greater than 4 DW 111: Reserved

1.1.4.3 Writeback Message to Recipient Thread

This message contains the byte or dwords data segment indicated in the message written to the GRF register offset indicated. Only the byte/dword(s) will be enabled, all other data in the GRF register is untouched.

1.1.5 GetTimeStamp Message [DevSNB+]

The GetTimeStamp message gives the ability for a requester thread to read the timestamps back from the message gateway. The message consists of a single 256-bit message payload.

AbsoluteTimeLap is based on an absolute wall clock in unit of nSec/uSec that is independent of context switch or GPU frequency adjustment. Message Gateway shares the same GPU timestamp. Details can be found in the TIMESTAMP register section in *vol1c Memory Interface and Command Stream*.

RelativeTimeLap is based on a relative time count that is counting the GPU clocks for the context. The relative time count is saved/restored during context switch.

1.1.5.1 Message Payload

DWord	Bit	Description
M0.7	31:0	**Reserved**
M0.6	31:0	**Reserved**
M0.5	31	**Return to High GRF**: 0: the return 128-bit data goes to the first half of the destination GRF register 1: the return 128-bit data goes to the second half of the destination GRF register
	30:8	Reserved : MBZ
	7:0	**Dispatch ID:** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion. This field is ignored by Message Gateway This field is only required for a thread that is created by a fixed function (therefore, not a child thread) and EOT bit is set for the message.
M0.4	31:0	Ignored
M0.3	31:0	Ignored
M0.2	31:0	Ignored
M0.1	31:0	Ignored
M0.0	31:0	Ignored

1.1.5.2 Writeback Message to Requester Thread

As the writeback message is only sent if the **AckReq** bit in the message descriptor is set, **AckReq** bit must be set for this message.

Only half of the destination GRF register is updated (via write-enables). The other half of the register is not changed. This is determined by the **Return to High GRF** control field.

Writeback Message if Return to High GRF is set to 0:

DWord	Bit	Description
W0.7:4		Reserved (not overwritten)
W0.3	31:0	**RelativeTimeLapHigh:** This field returns the MSBs of time lap for the relative clock since the previous reset. This field represents 1.024 uSec increment of the time stamp. Hardware handles the wraparound (over 64 bit boundary) of the timestamp. Format: U12
W0.2	31:20	**RelativeTimeLapLow:** This field returns the LSBs of time lap for the relative clock since the previous reset. This field represents 1/4 nSec increment of the time stamp. Hardware handles the wraparound (over 64 bit boundary) of the timestamp. Format: U12
	19:0	Reserved : MBZ
W0.1	31:0	**AbsoluteTimeLapHigh:** This field returns the MSBs of time lap for the absolute clock since the previous reset. This field represents 1.024 uSec increment of the time stamp. Hardware handles the wraparound (over 64 bit boundary) of the timestamp. Format: U12
W0.0	31:20	**AbsoluteTimeLapLow:** This field returns the LSBs of time lap for the absolute clock since the previous reset. This field represents 1/4 nSec increment of the time stamp. Hardware handles the wraparound (over 64 bit boundary) of the timestamp. Format: U12
	19:0	Reserved : MBZ

Writeback Message if Return to High GRF is set to 1:

DWord	Bit	Description
W0.7	31:0	**RelativeTimeLapHigh**
W0.6	31:20	**RelativeTimeLapLow**
	19:0	Reserved : MBZ
W0.5	31:0	**AbsoluteTimeLapHigh**
W0.4	31:20	**AbsoluteTimeLapLow**
	19:0	Reserved : MBZ
W0.3:0		Reserved : MBZ

1.1.6 BarrierMsg Message

The BarrierMsg message gives the ability for multiple threads to synchronize their progress. This is useful when there are data shared between threads. The message consists of a single 256-bit message payload.

Upon receiving one such message, Message Gateway increments the Barrier counter and mark the Barrier requester thread. There is no immediate response from the Message Gateway. When the counter value equates **Barrier Thread Count**, Message Gateway will send response back to all the Barrier requesters.

1.1.6.1 Message Payload

DWord	Bit	Description
M0.7	31:0	**Reserved**
M0.6	31:0	**Reserved**
M0.5	31:0	Ignored
M0.4	31:0	Ignored
M0.3	31:0	Ignored
M0.2	31:28	Ignored
	27:24	**BarrierID**. This field indicates which one from the 16 Barrier States is updated. Format: U4 Note: this field location matches with that of R0 header.
	23:16	**Barrier.Offset.** This is the offset for the Barrier to indicate the offset from the requester's RegBase. Regbase is 0 if Bypass Gateway Control is 1. Barrier.Offset + RegBase must be in the valid GRF range, but not point to r0. Otherwise, hardware behavior is undefined. It is in unit of 256-bit GRF register. The most significant bit of this field must be zero. Format = U8 Range = [0,127] [DevSNB] The least significant bit of this field is ignored by hardware, so only even offsets are allowed..
	15:0	Ignored
M0.1	31:0	Ignored
M0.0	31:4	Ignored

1.1.6.2 Writeback Message to Requester Thread

The writeback message is only sent if the **AckReq** bit in the message descriptor is set.

DWord	Bit	Description
W0.7:1		Reserved (not overwritten)
W0.0	31:20	Reserved
	19:16	**Shared Function ID:** Contains the message gateway's shared function ID.
	15:3	Reserved
	2:0	**Error Code** 000: *Successful*. No Error (Normal) 001: *Error (Barrier is inactive)*. Other encodings are reserved

1.1.6.3 Broadcast Writeback Message

[DevSNB] When the count for a Barrier (identified by BarrierID) reaches Barrier.Count, Message Gateway broadcasts the following messages to all threads using the BarrierID.

This message contains one single byte written to the GRF register at the *Barrier.Offset* + *RegBase* location. Only the byte will be enabled, all other data in the GRF register is untouched. In addition, the message will have the notification bit set for N0 so that the kernel set a dependency on either the register write or N0.

DWord	Bit	Description
W0.7:1		Reserved (not overwritten)
W0.0	31:8	Reserved (not overwritten)
	7:0	**[DevSNB] Barrier.Byte** If **Barrier Count Enable** was set on the barrier-completing BarrierMsg, this byte has a value of 0. If **Barrier Count Enable** was clear on the barrier-completing BarrierMsg, the value written is obtained from the Interface Descriptor. Format: U8

2. Unified Return Buffer (URB)

The Unified Return Buffer (URB) is a general-purpose buffer used for sending data between different threads, and, in some cases, between threads and fixed-function units (or vice-versa). A thread accesses the URB by sending messages.

2.1 URB Size

[DevSNB]: The GT1 product's URB provides 32KB of storage, arranged as 1024 256-bit *rows*. The GT2 product's URB provides 64KB of storage, arranged as 2048 256-bit *rows*. A row corresponds in size to an EU GRF register. Read/write access to the URB is generally supported on a row-granular basis.

A URB *entry* is a logical entity within the URB, referenced by an entry *handle* and comprised of some number of consecutive rows.

2.2 URB Access

The URB can be written by the following agents:

- Command Stream (CS) can write constant data into Constant URB Entries (CURBEs) as a result of processing CONSTANT_BUFFER commands.
- The Video Front End (VFE) fixed-function unit of the Media pipeline can write thread payload data in to its URB entries.
- The Vertex Fetch (VF) fixed-function unit of the 3D pipeline can write vertex data into its URB entries
- Threads can write data into URB entries via URB_WRITE messages sent to the URB shared function.

The URB can be read by the following agents:

- The Thread Dispatcher (TD) is the main source of URB reads. As a part of spawning a thread, pipeline fixed-functions provide the TD with a number of URB handles, read offsets, and lengths. The TD reads the specified data from the URB and provide that data in the thread payload pre-loaded into GRF registers.
- The Geometry Shader (GS) and Clipper (CLIP) fixed-function units of the 3D pipeline can read selected parts of URB entries to extract vertex data required by the pipeline.
- The Windower (WM) FF unit reads back depth coefficients from URB entries written by the Strip/Fan unit.

[DevSNB] Note that neither the CPU nor EU threads can read the URB directly.

2.3 State

The URB function is stateless, with all information required to perform a function being passed in the write message.

See URB Allocation (*Graphics Processing Engine*) for a discussion of how the URB is divided amongst the various fixed functions.

2.4 Messages [DevSNB]

There is only one type of message supported by the URB shared function: URB_WRITE. It is primarily used by a thread to write data in to an entry in the URB, as referenced by the passed handle. FF units of the 3D pipeline snoop these messages, and a side effect of the message may be some information being passed to the FF unit which spawned the thread.

This section documents the global aspects of the URB write messages. The actual data contained in the message differs for each fixed function – refer to *3D Pipeline* and the fixed-function chapters or details on 3D URB data formats, *Media* for media-specific URB data formats, and *Graphics Processing Engine* for details on Constant URB Entries (CURBEs).

Programming Notes:

- The **End of Thread** bit in the message descriptor may be set on URB messages only in threads dispatched by the vertex shader (VS), geometry shader (GS), clipper, and strips and fans (SF) units.
- The Kernel must do a write commit on the last write to DAP prior to a URB_WRITE with **End of Thread**. [DevSNB]

2.4.1 Execution Mask

The Execution Mask specified in the 'send' instruction determines which DWords within each message register are written to the URB.

2.4.2 Message Descriptor

Bit	Description
19	**Header Present** This bit must be set to **one** for all URB messages.
18:16	Ignored
15	**Complete** If clear, this signals that the URB entry(s) referenced by the handle(s) are <u>not yet</u> completely specified. This setting is used to perform partial writes to URB entries, as would be required when writing an entry larger than the maximum single message payload can accommodate. Only the final write would be marked "complete". Partial writes may be unordered. If set, this signals that there will be no further writes (past this one) to the specific URB entry(s) by the thread. A snooping FF unit uses this to identify when the corresponding URB entry(s) are completely specified, at which point the FF unit can initiate further operations the entry(s) (either a readback, passing the handle(s) down the pipeline, or immediate deallocation if the entry is "unused"). This bit is strictly control information passed to snooping FF units. The URB shared function itself does not use this bit for any purpose. Programming Notes: • The following message descriptor fields are only valid when **Complete** is set: **Used** • The following message header fields are only valid when **Complete** is set: **Handle 0 PrimType**, **Handle 0 PrimStart**, **Handle 0 PrimEnd**.

Bit	Description
14	**Used** If set, this signals that the URB entry(s) referenced by the handle(s) are valid outputs of the thread. In all likelihood this means that that entry(s) contains complete & valid data to be subject to further processing by the pipeline. If clear, this signals that the URB entry(s) referenced by the handle(s) are not valid outputs of the thread. Use of this setting will result in the handle(s) being immediately dereferenced by the owning FF unit. This setting is to be used by GS or CLIP threads to dereference handles it obtained (either in the initial thread payload or subsequent allocation writebacks) but subsequently determined were not required (e.g., the object was completely clipped out). **Programming Notes:** • Only GS and CLIP threads are permitted to utilize Used==0. All other threads are required (by design) to generate valid outputs in all cases. • This bit is strictly control information passed to snooping FF units. The URB shared function itself does not use this bit for any purpose. • This bit is only valid when **Complete** is set, i.e., it is ignored on partial writes.
13	**Allocate** If set, this requests that an additional destination URB entry be allocated to the thread by the spawning FF unit. The FF unit will return the handle to this URB entry via a message writeback operation in response to this message (see writeback format below). Therefore, threads must specify a writeback register in 'send' instructions issuing messages with this bit set. If clear, an additional allocation is not requested. **Programming Notes:** • This bit is strictly control information passed to snooping FF units. The URB shared function itself does not use this bit for any purpose. • This bit is valid on all URB_WRITE messages, e.g., it could be used to allocate a new handle on a partial write (**Complete** not set). • Only one Allocate request (per thread) can be outstanding. Upon requesting an allocation, the thread must wait for the handle to be returned (written back) before another allocation can be requested.
12	**Fast Composite Restriction Check Pass** If set on the end of thread message, this field indicates that the setup kernel portion of the fast composite restriction check has passed. This field is ignored for threads dispatched by units other than Strips and Fans. This field is also ignored unless at least one contiguous dispatch mode is enabled *and* at least one normal dispatch mode is enabled in WM_STATE.
11:10	**Swizzle Control.** This field is used to specify which "swizzle" operation is to be performed on the write data. It indirectly specifies whether one or two handles are valid. 00: URB_NOSWIZZLE The message data is to be written directly to a single URB entry (Handle 0). 01: URB_INTERLEAVED The message contains data to be written to two URB entries. The message data provided is interleaved such that the upper DWords (7:4) of each 256-bit unit contain data to be written to Handle 1, and the lower DWords (3:0) contain data to be written to Handle 0. The URB shared function will de-interleave this data and write the two separate data streams to the two entries using the single Offset value (see Offset below for more details). 10: URB_TRANSPOSE This message contains data that is to be "transposed" before being written to the URB. The transpose applied is tailored to the passing of data between the SF and WM stages – it is not a generic transpose. (See description below). Therefore, the assumption is that this mode will

Bit	Description
	only be used by Setup threads, where the setup-result data is swizzled before being written to the URB in order to provide a more optimal format for use in a subsequent PS thread. (See Strip/Fan, Windower chapters). If **Transposed URB Read Enable** (WM_STATE) is set, the Setup thread must use URB_NOSWIZZLE to write the coefficient data (it will be transposed whenever the URB is read). URB_TRANSPOSE must only be used when **Transposed URB Read Enable** is clear. See Programming Restrictions in the URB_TRANSPOSE subsection below. 11: Reserved
9:4	**Offset.** This field specifies a destination offset (in 256-bit units) from the start of the URB entry(s), as referenced by **URB Return Handle *n***, at which the data (if any) will be written. When URB_INTERLEAVED is used, this field provides a 256-bit granular offset applied to both URB entry destinations. When URB_TRANSPOSE is used, this field provides a 256-bit granular offset applied to the URB entry destination. The least significant bit of **Offset** must be zero.
3:0	**URB Opcode** 0: URB_WRITE 1: FF_SYNC all other codes are Reserved

The following table lists the valid and invalid combinations of the Complete, Used, Allocate and EOT bits:

Complete	Used	Allocate	EOT	Valid?	Usage
0	d/c	0	0	Valid:	Normal partial-write or non-write of URB.
0	d/c	0	1	Valid only if any and all preceding URB entries have been marked as “complete” and there is no outstanding Allocate request.	Thread terminate w/ non-write of URB
0	d/c	1	0	Valid only if any and all preceding URB entries have been marked as “complete” and there is no outstanding Allocate request.	Non-write of URB with request for an additional handle.
0/1	d/c	1	1	Invalid. Thread must never terminate with an outstanding writeback request.	n/a
1	0	0/1	0	Valid	Dereference of URB entry without/with new allocation request.
1	0	0	1	Valid	Dereference of URB entry and thread termination.
1	1	0/1	0	Valid	Completion of URB entry output without/with new allocation request.

Complete	Used	Allocate	EOT	Valid?	Usage
1	1	0	1	Valid	Completion of URB entry output and thread termination.

2.4.3 URB_WRITE

2.4.3.1 URB_WRITE Message Header

DWord	Bit	Description
M0.7	31:0	**Reserved**
M0.6	31:0	**Reserved**
M0.5	31:8	Ignored
	7:0	**FFTID.** The Fixed Function Thread ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion.
M0.4	31:0	Ignored
M0.3	31:0	Ignored
M0.2	31:27	Ignored
	26:16	**SONumPrimsWritten Increment Value**. **[DevSNB+]** This field is ignored if the **SVBI Post Increment** field is set in the GS_3DSTATE in the 3D pipeline chapter.
	15:8	Ignored
	7	Ignored
	6:2	**Handle 0 PrimType.** This field associates a primitive type with the vertex written at Handle 0. **NOTE:** This field is only defined when the GS or Clipper FF unit is the target FF unit. Otherwise it is Reserved:MBZ.
	1	**Handle 0 PrimStart.** This field is used to indicate that the vertex written at Handle 0 is the first vertex of a primitive. NOTE: This field is only defined when the GS or Clipper FF unit is the target FF unit. Otherwise it is Reserved:MBZ.
	0	**Handle 0 PrimEnd.** This field is used to indicate that the vertex written at Handle 0 is the last vertex of a primitive. NOTE: This field is only defined when the GS or Clipper FF unit is the target FF unit. Otherwise it is Reserved:MBZ.
M0.1	31:16	**Handle ID 1.** This ID is assigned by the fixed function unit and links the work in channel 1 to a specific entry within the fixed function unit. This field is ignored unless **Swizzle Control** indicates Interleave mode.
	15:0	**URB Return Handle 1.** This is the URB handle where channel 1's results are to be placed. This field is ignored unless **Swizzle Control** indicates interleave mode.

DWord	Bit	Description
M0.0	31:16	**Handle ID 0.** This ID is assigned by the fixed function unit and links the work in channel 0 to a specific entry within the fixed function unit.
	15:0	**URB Return Handle 0.** This is the URB handle where channel 0's results are to be placed.

2.4.3.2 URB_WRITE Message Payload

For the URB message, the message payload will be written to the URB entries indicated by the URB return handles in the message header.

While GS and CLIP threads will write one vertex at a time to the URB, the VS will write two interleaved vertices. The description of the URB write messages will refer to the per-vertex DWords described in the Vertex URB Entry Formats section of the *3D Overview* chapter.

Payload	Usage
URB_NOSWIZZLE	The message payload contains data to be written to a single URB entry (e.g., one Vertex URB entry). The **Swizzle Control** field of the message descriptor must be set to 'NoSwizzle'.
URB_INTERLEAVED	The message payload contains data to be written to two separate URB entries. The payload data is provided in a high/low interleaved fashion. The **Swizzle Control** field of the message descriptor must be set to 'Interleave'.
URB_TRANSPOSE	The message payload contains data that is to be transposed before being written to the URB. See the *Strip & Fan (SF) Unit* chapter for details on the source and destination data layouts and intended usage model.

2.4.3.2.1 URB_NOSWIZZLE

URB_NOSWIZZLE is used to simply write data into consecutive URB locations (no data swizzling or transposition applied).

Programming Notes:

- The URB function will ignore the Channel Enables associated with this message and write all channels into the URB.
- When **Transposed URB Read Enable** (WM_STATE) is set, the Setup thread must use URB_NOSWIZZLE to write the coefficient data into the URB (it will be transposed whenever the URB is read).

When URB_NOSWIZZLE is used to write vertex data, the following table shows an example layout of a URB_NOSWIZZLE payload containing one (non-interleaved) vertex containing *n* pairs of 4-DWord vertex elements (where for the example, *n* is >2).

DWord	Bit	Description
M1.7	31:0	**Vertex Data [7]**
M1.6	31:0	**Vertex Data [6]**
M1.5	31:0	**Vertex Data [5]**
M1.4	31:0	**Vertex Data [4]**
M1.3	31:0	**Vertex Data [3]**
M1.2	31:0	**Vertex Data [2]**
M1.1	31:0	**Vertex Data [1]**
M1.0	31:0	**Vertex Data [0]**
M2.7	31:0	**Vertex Data [15]**
M2.6	31:0	**Vertex Data [14]**
M2.5	31:0	**Vertex Data [13]**
M2.4	31:0	**Vertex Data [12]**
M2.3	31:0	**Vertex Data [11]**
M2.2	31:0	**Vertex Data [10]**
M2.1	31:0	**Vertex Data [9]**
M2.0	31:0	**Vertex Data [8]**
…		**…**
Mn.7	31:0	**Vertex Data [8(n-1)+7]**
Mn.6	31:0	**Vertex Data [8(n-1)+6]**
Mn.5	31:0	**Vertex Data [8(n-1)+5]**
Mn.4	31:0	**Vertex Data [8(n-1)+4]**
Mn.3	31:0	**Vertex Data [8(n-1)+3]**
Mn.2	31:0	**Vertex Data [8(n-1)+2]**

DWord	Bit	Description
Mn.1	31:0	**Vertex Data [8(n-1)+1]**
Mn.0	31:0	**Vertex Data [8(n-1)+0]**

2.4.3.2.2 URB_INTERLEAVED

The following table shows an example layout of a URB_INTERLEAVED payload containing two interleaved vertices, each containing *n* 4-DWord vertex elements (n>1).

Programming Restrictions:

- At least 256 bits per vertex (512 bits total, M1 & M2) must be written. Writing only 128 bits per vertex (256 bits total, M1 only) results in UNDEFINED operation.
- The URB function will use (not ignore) the Channel Enables associated with this message.
- **[DevSNB]** Interleave writes must be in multiples of 256 per vertex.

DWord	Bit	Description
M1.7	31:0	**Vertex 1 Data [3]**
M1.6	31:0	**Vertex 1 Data [2]**
M1.5	31:0	**Vertex 1 Data [1]**
M1.4	31:0	**Vertex 1 Data [0]**
M1.3	31:0	**Vertex 0 Data [3]**
M1.2	31:0	**Vertex 0 Data [2]**
M1.1	31:0	**Vertex 0 Data [1]**
M1.0	31:0	**Vertex 0 Data [0]**
M2.7	31:0	**Vertex 1 Data [7]**
M2.6	31:0	**Vertex 1 Data [6]**
M2.5	31:0	**Vertex 1 Data [5]**
M2.4	31:0	**Vertex 1 Data [4]**
M2.3	31:0	**Vertex 0 Data [7]**
M2.2	31:0	**Vertex 0 Data [6]**
M2.1	31:0	**Vertex 0 Data [5]**
M2.0	31:0	**Vertex 0 Data [4]**
…		**…**
Mn.7	31:0	**Vertex 1 Data [4(n-1)+3]**
Mn.6	31:0	**Vertex 1 Data [4(n-1)+2]**
Mn.5	31:0	**Vertex 1 Data [4(n-1)+1]**
Mn.4	31:0	**Vertex 1 Data [4(n-1)+0]**

DWord	Bit	Description
Mn.3	31:0	**Vertex 0 Data [4(n-1)+3]**
Mn.2	31:0	**Vertex 0 Data [4(n-1)+2]**
Mn.1	31:0	**Vertex 0 Data [4(n-1)+1]**
Mn.0	31:0	**Vertex 0 Data [4(n-1)+0]**

2.4.3.2.3 URB_TRANSPOSE

The following table shows an example layout of a URB_TRANSPOSE payload and how the data is transposed and stored in the destination URB entry. Note that Source Row 0, Source Row 1, and implied row of all-zero, and Source Row 3 is transposed and stored in successive 4-DW locations in the destination. This is then repeated for the next 3 rows of the source payload. For the intended usage model in the Setup thread, Source Row 0 would contain "Cx" coefficients for the first 8 attributes, Source Row 1 would contain "Cy" coefficients for the first 8 attributes, and Source Row 2 would contain "C0" coefficients for the first 8 attributes, then repeating for the next 8 attributes. Insertion of the implied all-zero row is required to align the Cx,Cy and C0 attributes into half-rows within the URB. This permits the used of the "LINE" instruction to initiate attribute interpolation in the subsequent PS thread.

Programming Notes:

- The message payload must contain a multiple of 3 Source Rows of data (excluding the message header).
- The URB function will ignore the Channel Enables associated with this message and write all channels into the URB.
- When **Transposed URB Read Enable** (WM_STATE) is set, the Setup thread must use URB_NOSWIZZLE to write the coefficient data into the URB (it will be transposed whenever the URB is read). URB_TRANSPOSE should only be used when **Transposed URB Read Enable** is clear.

Table 2-1. URB_TRANSPOSE Payload

DWord	Bit	Description
M1.0-7	31:0	**Source Row 0 (e.g., Cx coeffs for the 1st set of 8 attributes)**
M2.0-7	31:0	**Source Row 1 (e.g., Cy coeffs for the 1st set of 8 attributes)**
M3.0-7	31:0	**Source Row 2 (e.g., C0 coeffs for the 1st set of 8 attributes)**
M4.0-7	31:0	**Source Row 3 (e.g., Cx coeffs for the 2nd set of 8 attributes)**
M5.0-7	31:0	**Source Row 4 (e.g., Cy coeffs for the 2nd set of 8 attributes)**
M6.0-7	31:0	**Source Row 5 (e.g., C0 coeffs for the 2nd set of 8 attributes)**
...	31:0	**...**

Table 2-2.URB_TRANSPOSE URB Destination Layout

	URB DW							
URB Row	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
n+0	M3.1	0	M2.1	M1.1	M3.0	0	M2.0	M1.0
n+1	M3.3	0	M2.3	M1.3	M3.2	0	M2.2	M1.2
n+2	M3.5	0	M2.5	M1.5	M3.4	0	M2.4	M1.4
n+3	M3.7	0	M2.7	M1.7	M3.6	0	M2.6	M1.6
n+4	M6.1	0	M5.1	M4.1	M6.0	0	M5.0	M4.0
n+5	M6.3	0	M5.3	M4.3	M6.2	0	M5.2	M4.2
n+6	M6.5	0	M5.5	M4.5	M6.4	0	M5.4	M4.4
n+7	M6.7	0	M5.7	M4.7	M6.6	0	M5.6	M4.6
	...	...	...	...	...	...	...	...

2.4.3.3 Writeback Message for URB Entry Allocate

A writeback only occurs if the **Allocate** bit is set in the message descriptor. A single register is returned containing the URB Return Handle and Handle ID for the allocated handle in the low DWord is returned. All high DWords contain zero.

DWord	Bit	Description
W0.7:1		Reserved : MBZ
W0.0	31:16	**Handle ID.** This ID is assigned by the fixed function unit and links the thread to a specific entry within the fixed function unit.
	15:0	**URB Return Handle.** This is the URB handle where the thread's results are to be placed.

2.4.4 FF_SYNC Messages

The FF_SYNC message is used to pass critical information between GS/Clip threads and the GS/Clip FF units, as well as providing Gs/Clip thread synchronization (ordering). GS threads report various counts resulting from running the GS and/or SO functions, prior to performing any output (to SOB buffers or to URB handles). Clip threads report only # of handles required. A message response (writeback) length of 1 GRF will be indicated on the 'send' instruction if the thread requires response data and/or synchronization. Refer to the GS/Clip stage chapter for details.

2.4.4.1 FF_SYNC Message Header

Dword	Bit	Description
M0.7	31:0	**Reserved**
M0.6	31:0	**Reserved**
M0.5	31:8	Ignored
	7:0	**FFTID.** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion.
M0.4	31:0	Ignored
M0.3	31:0	Ignored
M0.2	31:7	Ignored
M0.1	31:16	Ignored
	15:0	**(GS-only) NumGSPrimsGenerated**. The number of objects (e.g., triangles) generated by the GS function performed by the thread. If the GS function is not enabled, this field MBZ. Format: U16 Range: [0,1024]
M0.0	31:16	**(GS-only) NumSOVertsToWrite**. The number of (expanded-to-list) vertices generated by the SO function performed by the thread. This represents the number of vertices the thread will attempt to write to the SOB(s) in memory, once it obtains the SVBI(s) in the FF_SYNC writeback. Note that overflow may occur either (a) prior to the SVBI(s) are returned in the writeback or (b) in the process of this thread outputting to the SOBs. In either case, the thread needs to check for overflow once it receives the writeback, based on the returned SVBI(s) and the number of vertices it must attempt to output. If the SO function is not enabled, this field MBZ. **[DevSNB+]** This field is ignored if the **SVBI Post Increment** field is set in the GS_3DSTATE in the 3D pipeline chapter. Format: U16 Range: [0,3066] (1024-vertex tristrip = 1022 triangles = 3066 trilist vertices)
	15:0	**(GS-only) NumSOPrimsNeeded**. The number of objects (e.g., triangles within a trillist) generated by the SO function performed by the thread (exclusive of any SOB overflow). If the SO function is not enabled, this field MBZ. **[DevSNB+]** This field is ignored if the **SVBI Post Increment** field is set in the GS_3DSTATE in the 3D pipeline chapter. Format: U16 Range: [0,1024]

2.4.4.2 FF_SYNC Writeback Message

(Both GS & Clip): DWord W0.0 of the writeback data contains initial handle information. If **Handle Valid** is clear, the FF unit did not have a handle available to be allocated as the initial handle – the thread will need to use **the URB_WRITE message to obtain the initial handle. Otherwise the Handle ID and URB** Return Handle fields are valid and can be used to write the first VUE.

(GS-only) The writeback data contains the SVBI values to be used as starting write indices by the GS thread. It is the responsibility of the GS thread to perform SOB overflow processing. If the GS thread is

not performing StreamOutput and was simply using the writeback to provide GS vertex output synchronization, the return data is to be ignored.

(Clip-only) Dwords W0.1-7 of the writeback data are ignored.

DWord	Bit	Description
W0.7: W0.5	31:0	Reserved
W0.4	31:0	**(GS-only) Streamed Vertex Buffer Index 3** This field represents the value of SVBI[3] that is to be used as the starting index for the GS thread. If the thread is not performing StreamOutput, this field is ignored. Format = U32 Range = $[0,2^{27}-1]$
W0.3	31:0	**(GS-only) Streamed Vertex Buffer Index 2**
W0.2	31:0	**(GS-only) Streamed Vertex Buffer Index 1**
W0.1	31:0	**(GS-only) Streamed Vertex Buffer Index 0**
W0.0	31:24	Reserved
	23:16	**Handle ID.** This ID is assigned by the FF unit and links the thread to a specific entry within the FF unit. Format: Reserved for Implementation Use
	15:12	Reserved
	11:0	**URB Return Handle.** This is the initial destination URB handle passed to the thread. If the thread does output URB entries, this identifies the first destination URB entry. **[DevSNB+]** Format: U12 256 bit URB Handle Address

3. Execution Unit ISA

3.1 Introduction

3.1.1 Objective and Scope

The core architecture consists of an array of multi-threaded processors, also referred to as Execution Units (EU). This Instruction Set Architecture (ISA) document specifies the instructions executable on the EUs of the architecture. It defines the data types in the GEN architecture. It includes the binary format (machine code) and ASCII format (native syntax) of each instruction. It also provides example usages of instructions and modes of instructions, and certain data formats. The programming guideline in appendix provides information to help developers to understand the usage of ISA. However, it is not intended to be a comprehensive tutorial.

3.1.2 Terms and Acronyms

AIP	Application IP. This is part of the control registers for exception handling for a thread. Upon an exception, hardware moves the current IP into this register and then jumps to SIP.
ARF	Architecture Register File. It is a collection of architecturally visible registers for a thread such as address registers, accumulator, flags, notification registers, IP, null, etc. ARF should not be mistaken as just the address registers.
B	Byte. As a numerical data type of 8 bits, B represents a signed byte integer. It is used to specify the type of an operand in an instruction.
BNF	Backus Naur Form, a formal notation to describe the syntax of a given language. The meta symbols of BNF include "**::=**", "**|**", and "**< >**", where "::=" means "is defined as"; "|" means "or"; and angle brackets "<" and ">" are used to surround category names.
CR	Control Register. These read-write registers are used for thread mode control and exception handling for a thread.
D	Double word (DWord). As a fundamental data type, D or DW represents 4 bytes. It may be used to specify the type of an operand in an instruction.
EOT	End Of Thread. This is a message sideband signal on the Output message bus signifying that the message requester thread is terminated. A thread must have at least one SEND instruction with the EOT bit in the message descriptor field set in order to properly terminate.
EU	Execution Unit. An EU is a multi-threaded processor within the multi-processor system. Each EU is a fully-capable processor containing instruction fetch and decode, register files, source operand swizzle and SIMD ALU, etc. An EU is also referred to as a Core.

EUID	Execution Unit Identifier. The 4-bit field within a thread state register (SR0) that identifies the row and column location of the EU where a thread is located. A thread can be uniquely identified by the EUID and TID.
ExecSize	Execution Size.
Execution Size	Execution Size indicates the number of data elements processed by a SIMD instruction. It is one instruction field and can be changed at a per instruction level.
FLT_MAX	The magnitude of the maximum represent-able single-precision floating number according to IEEE-754 standard. FLT_MAX has an exponent of 0xFE and a mantissa of all one's.
fmax	Same as FLT_MAX.
Core	Alternative name for an EU in the multi-processor system.
GRF	General Register File. This is the most commonly used read-write register space organized as an array of 256-bit registers for a thread.
ISA	Instruction Set Architecture. The ISA describes the instructions supported by an EU. A sequence of instructions forms a thread executed on an EU.
JIT	Just-In-Time compiler
LSB	Least Significant Bit
Message	Messages are data packages transmitted from a thread to another thread, to another shared function or to another fixed function. Message passing is the primary communication mechanism of the architecture.
MRF	Message Register File. This is the write-only register space, organized as an array of 256-bit registers, for a thread to communicate with shared functions or other threads.
MSB	Most Significant Bit
DQ	Double Quad word (DQword). As a fundamental data type, DQ represents 16 bytes.
POR	Plan Of Record
QW	Quad Word (QWord). As a fundamental data type, QW represents 8 bytes.
QQ	Quad Quad word (QQword). As a fundamental data type, QQ represents 32 bytes.
Sub-Register	Subfield of a SIMD register. A SIMD register is an aligned fixed size register for a register file or a register type. For example, a GRF register, *r2*, is a 256-bit wide, 256-bit aligned register. A sub-register, *r2.3:d*, is the fourth dword of GRF register *r2*.
SIMD	Single Instruction Multiple Data. The term SIMD can be used to describe the kind of parallel processing architecture that exploits data parallelism at the instruction level. It can also be used to describe the instructions in such an architecture.

SIP	System IP. There is one global System IP register for all the threads. From a thread's point of view, this is a virtual read-only register. Upon an exception, hardware performs certain book-keeping functions and then jumps to SIP.
SR	State Register. The read-only registers containing the state information of the current thread, including the EUID/TID, Dispatcher Mask, and System IP.
Thread	A thread is an instance of a kernel program executed on an EU. The life cycle for a thread starts from the executing the first instruction after being dispatched from Thread Dispatcher to an EU to the execution of the last instruction – a send instruction with EOT that signals the thread termination. Threads in the system may be independent from each other or communicate with each other through the Message Gateway share function.
TID	Thread Identifier. The 2-bit field within a thread state register (SR0) that identifies which out of the four possible thread slots on the EU is executing that thread. A thread can be uniquely identified by the EUID and TID.
TS	Thread Spawner. TS is the second and the last fixed function stage of the media pipeline.
V	Immediate integer vector. As a numerical data type of 32 bits, an immediate integer vector of type V contains 8 signed integer elements with 4 bits each. The 4-bit integer element is in 2's compliment form. It may be used to specify the type of an immediate operand in an instruction.
VF	Immediate floating point vector. As a numerical data type of 32 bits, an immediate floating point vector of type VF contains 4 floating point elements with 8-bit each. The 8-bit floating point element contains a sign field, a 3-bit exponent field and a 4-bit mantissa field. It may be used to specify the type of an immediate operand in an instruction.
W	Word. As a numerical data type of 16 bits, W represents a signed word integer. It is used to specify the type of an operand in an instruction.
URB	Unified Return Buffer. The on-chip memory managed/shared by Fixed Functions. Threads use the URB to return data that will be consumed either by a Fixed Function or other threads.
UB	Unsigned Byte integer. A numerical data type of 8 bits. It may be used to specify the type of an operand in an instruction.
UD	Unsigned Double Word integer. A numerical data type of 32 bits. It may be used to specify the type of an operand in an instruction.
UW	Unsigned Word integer. A numerical data type of 16 bits. It may be used to specify the type of an operand in an instruction.
VFE	Video Front End. VFE is the first fixed function stage of the media pipeline.

3.1.3 Formats and Conventions

In order to conveniently (and without ambiguity) describe the register files with 256-bit wide registers that may contain various data types with different data element widths, it is important to use a consistent table format to represent the registers. Throughout this document, we will adopt the following table formats and conventions. When a register or a number is presented by a row, increasing order is always **from right to left** and then **top down** pictorially. In other words, for a bit field, the LSB to MSB is from right to left; for a byte sequence, the least significant byte to the most significant byte is also from right to left. This is consistent with the 'Little Endian' convention used by IA-32 machines. The following tables depict the layout formats for different data units.

7	6	5	4	3	2	1	0	**Bits**
								A Byte

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	**Bits**
Byte 1								Byte 0								**A Word**

31	24	23	16	15	8	7	0	**Bits**
Byte 4		Byte 2		Byte 1		Byte 0		**A DWord**

31	30	29	..																									3	2	1	0	**32 Bytes**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	**16 Words**

7	6	5	4	3	2	1	0	**8 DWords**

7	6	5	4	3	2	1	0	**16 DWords**
15	14	13	12	11	10	9	8	

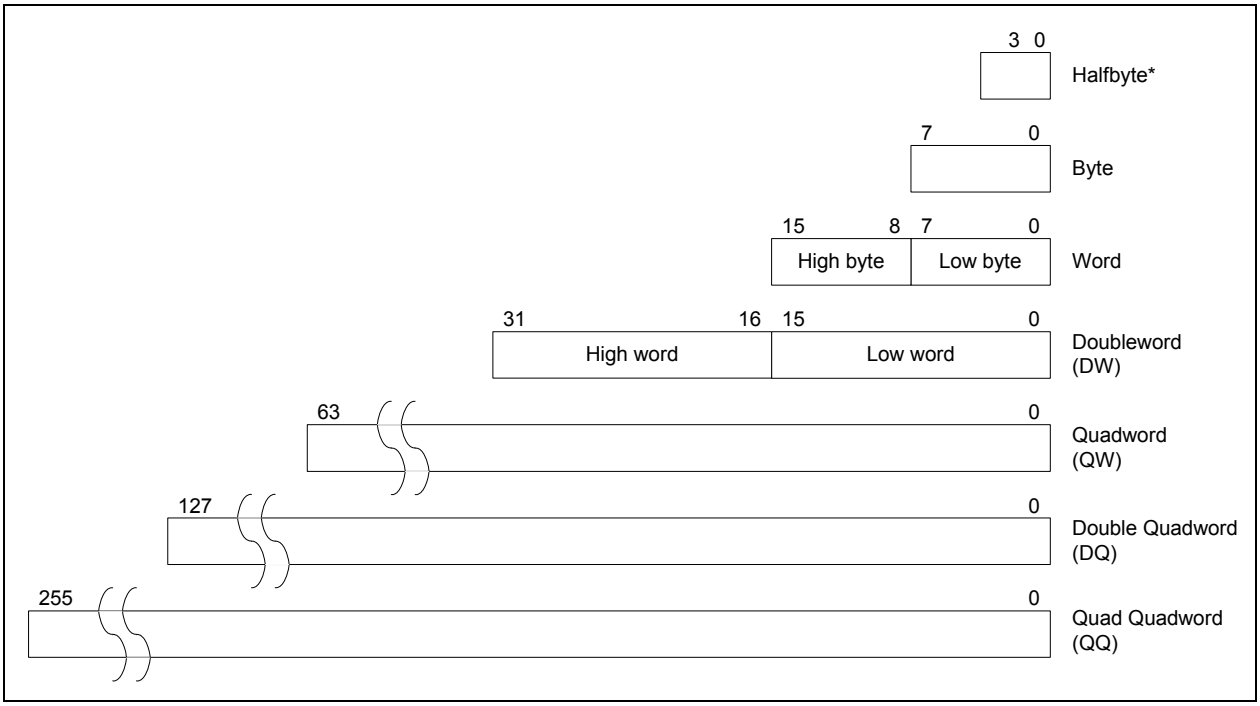
With this convention, we note that the execution channels are logically viewed as from right to left too, which is a little bit unconventional. However, as shown in the *GEN Execution Environment* Chapter, it matches with the bit order of the flag registers. This also impacts the view of a GRF register region, now the region origin is located at the upper-right corner and a region row is viewed from right to left.

4. EU Data Types

4.1 Fundamental Data Types

The fundamental data types in the architecture are halfbyte, byte, word, doubleword (DW), quadword (QW), double quadword (DQ) and quad quadword (QQ). They are defined based on the number of bits of the data type, ranging from 4 bits to 256 bits. As shown in **Error! Reference source not found.**, a halfbyte contains 4 bits, a byte contains 8 bits, a word contains two bytes, and a doubleword (dword) contains two words, and so on. Halfbyte is a special data type such that it cannot be accessed directly as standalone data element. It is only allowed as a subfield of the numerical data type of "packed signed halfbyte integer vector" described in the next section.

Figure 4-1 Fundamental data types



With the exception of halfbyte, the access of a data element to/from a register or to/from memory must be aligned on the natural boundaries of the data type. The natural boundary for a word has an even-numbered address in unit of byte. The natural boundary for a doubleword has an address divisible by 4 bytes. Similarly, the natural boundary for a quadword, double quadword and quad quadword has an address divisible by 8, 16, and 32 bytes, respectively. Quadword, double quadword and quad quadword do not have corresponding numerical data type. Instead, they are used to describe a group (a vector) of numerical data elements of smaller size align to larger natural boundaries.

4.2 Numerical Data Types

The numerical data types defined in the architecture include signed and unsigned integers and floating-point numbers (floats) of various numbers of bits. The following illustration details the notation, size and numerical range of each data type. The largest numerical data type has 32 bits.

Figure 4-2 Integer numerical data types

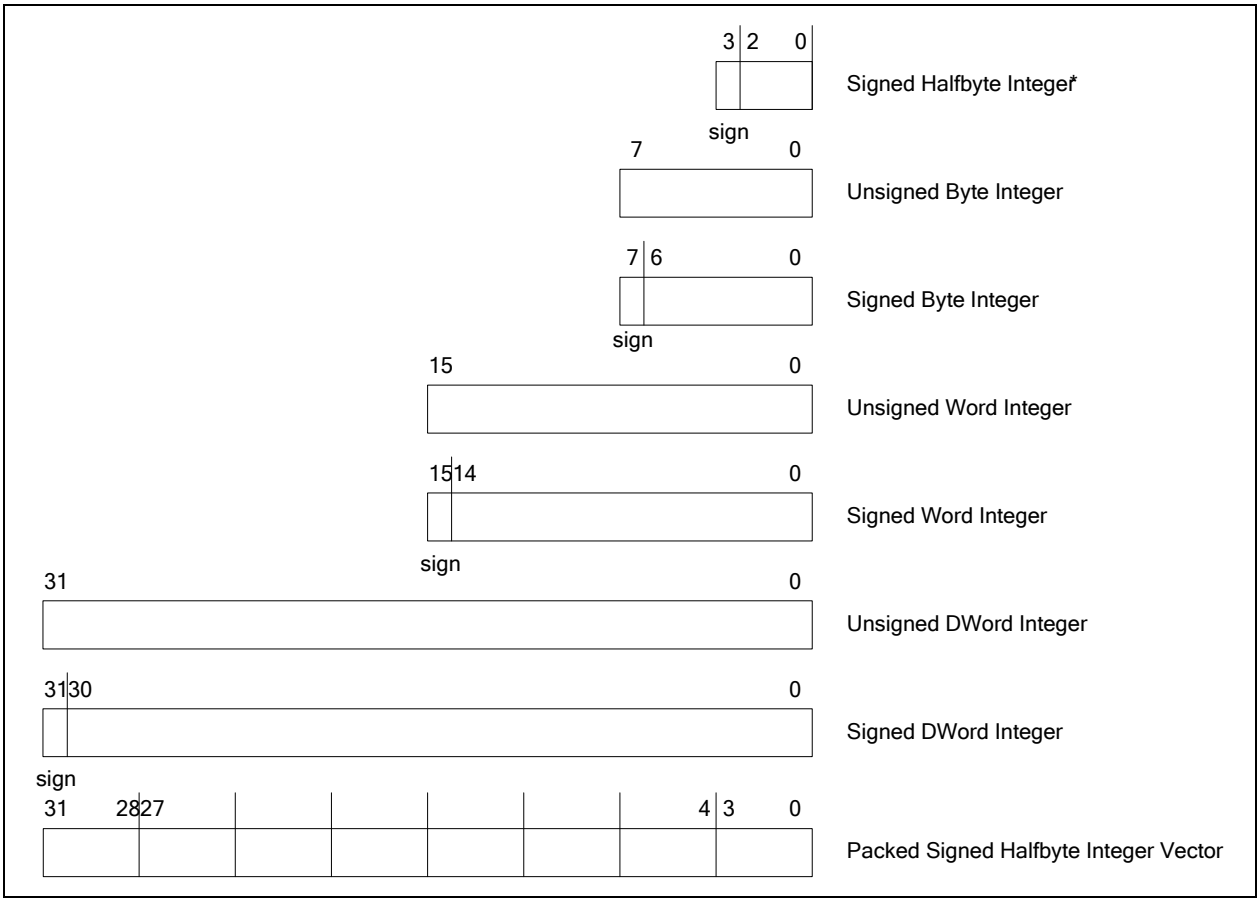


Figure 4-3. Floating point numerical data types

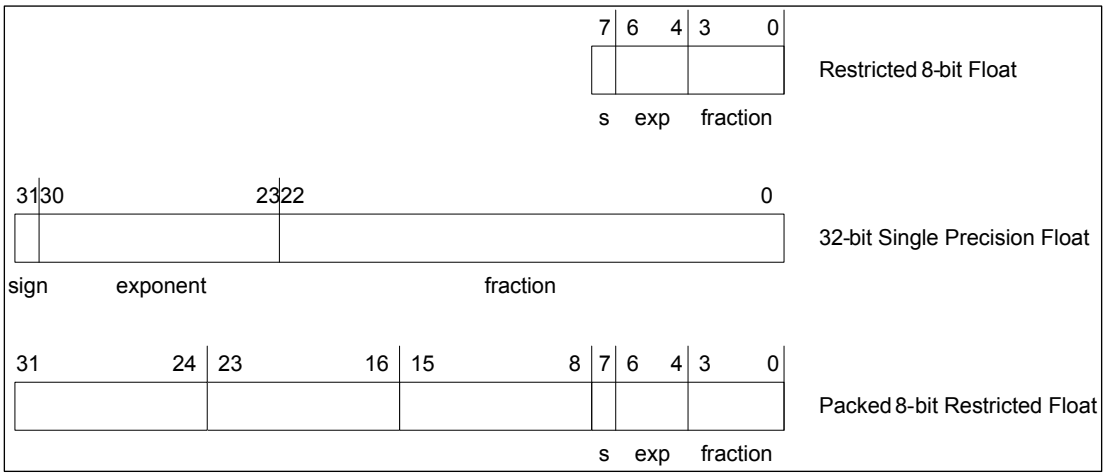


Table 4-1. Formats and ranges of numerical data types

Notation	Numerical Data Types	Fundamental Data Type	Range
UB	Unsigned Byte Integer	Byte	[0, 255]
B	Signed Byte Integer	Byte	[-128, 127]
UW	Unsigned Word Integer	Word	[0, 65535]
W	Signed Word Integer	Word	[-32768, 32767]
UD	Unsigned Doubleword Integer	Doubleword	$[0, 2^{32} – 1]$
D	Signed Doubleword Integer	Doubleword	$[–2^{31}, 2^{31} – 1]$
F	Single Precision Float	Doubleword	$[–(2–2^{-23})^{127}…–2^{-149}, 0.0, 2^{-149}… (2–2^{-23})^{127}]$
n/a	Signed Halfbyte Integer	Halfbyte	[–8, 7]
V	Packed Signed Halfbyte Integer Vector	Doubleword	[–8, 7]
n/a	Restricted 8-bit Float	Byte	[–31…–0.125, 0, 0.125… 31]
VF	Packed Restricted Float Vector	Doubleword	[–31…–0.125, 0, 0.125… 31]

4.2.1 Unsigned Integers

Unsigned integers are unsigned binary numbers contained in a byte, a word or a doubleword. The range for an unsigned byte integer is from 0 to 255. The range for an unsigned word integer is from 0 to 65535. The range for an unsigned doubleword integer is from 0 to $2^{32} – 1$.

The short hand notation for an unsigned byte integer, an unsigned word integer, and an unsigned doubleword integer is **UB**, **UW**, **UD**, respectively.

4.2.2 Signed Integers

Signed integers are signed binary number in 2's compliment form contained in a halfbyte, a byte, a word or a doubleword. A signed halfbyte integer has a numerical range from –8 to 7 with the sign bit at bit 3. A signed byte integer has a range from –128 to 127 with the sign at bit 7. A signed word integer is has a range from -32768 to 32767 with the sign at bit 15. A signed doubleword integer has a range from $–2^{31}$ to $2^{31} – 1$ with the sign at bit 31.

The short hand notation for a signed byte integer, a signed word integer, and a signed doubleword integer is **B**, **W**, **D**, respectively.

4.2.3 Single Precision Floating-Point Numbers

The single precision floating point numbers is contained in a doubleword. Floating point format is as defined in IEEE Standard 754 for Binary Floating-Point Arithmetic. Maximal representable number is $(2–2^{-23})^{127}$ and the minimal number is $– (2–2^{-23})^{127}$. The smallest fractional negative number $–2^{-149}$ and the smallest fractional positive number is 2^{-149}. Value 0.0 has no fractional parts.

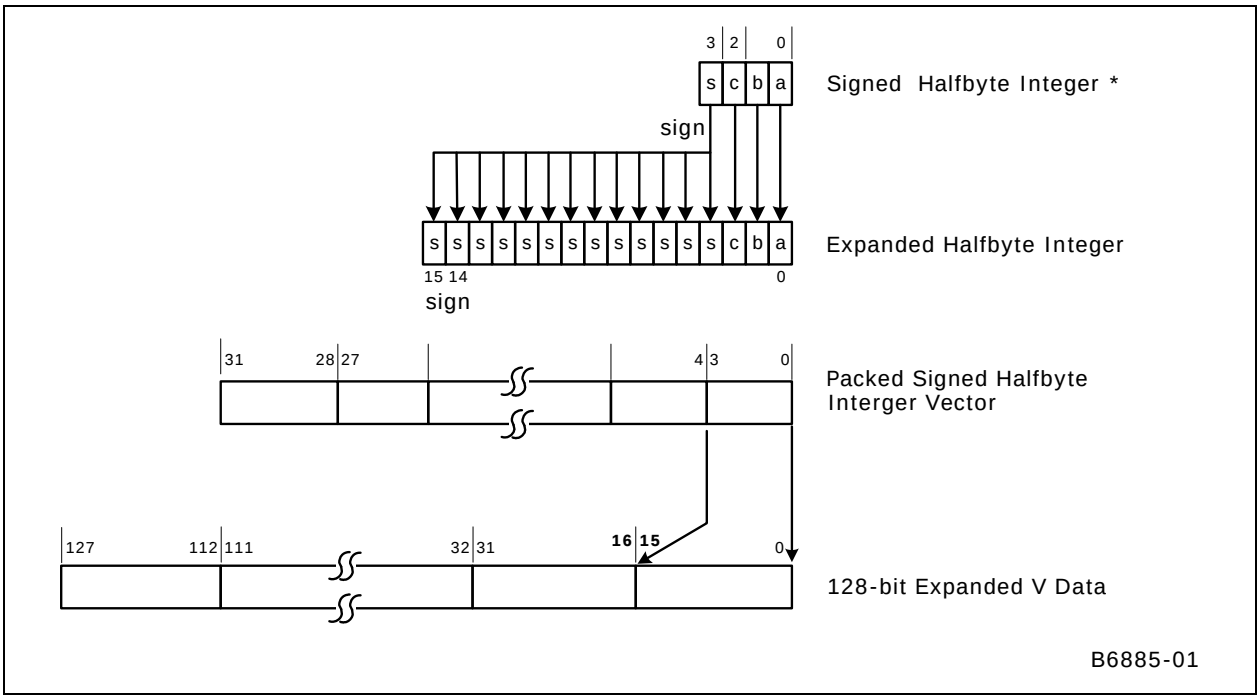
The short hand format notation for a single precision floating-point number is **F**.

4.2.4 Packed Signed Half-Byte Integer Vector

A packed signed halfbyte integer vector consists of 8 signed halfbyte integers contained in a doubleword. Each signed halfbyte integer element has a range from -8 to 7 with the sign on bit 3. This numerical data type is only used by an immediate source operand of doubleword in an instruction. It cannot be used for the destination operand or a non-immediate source operand. The hardware converts the 32-bit vector into 8-element signed word vector by sign extension. This is illustrated in Figure 4-4.

The short hand format notation for a packed signed half-byte vector is **V**.

Figure 4-4. Converting a packed half-byte vector to a 128-bit signed integer vector



4.2.5 Packed 8-bit Restricted Float Vector

A packed restricted float vector consists of 4 8-bit restricted floats contained in a doubleword. Each restricted float has the sign at bit 7, a 3-bit coded exponent in bits 4 to 6, a 4-bit fraction in bits 0 to 3, and an implied integer 1. The exponent is in excess-3 format – having a bias of 3. Restricted float provides zero, positive/negative normalized numbers with a small range (3-bit exponent) and small precision (4-bit fraction). This numerical data type is only used by an immediate source operand of doubleword in an instruction. It cannot be used for the destination operand, or a non-immediate source operand.

Figure 4-5 shows how to convert an 8-bit restricted float into a single precision float. Converting a 3-bit exponent with a bias of 3 to an 8-bit exponent with a bias of 127 is by adding 4, or equivalently copying bit 2 to bit 7 and putting the inverted bit 2 to bits 6:2. A special logic is also needed to take care of positive/negative zeros.

Figure 4-5. Conversion from a Restricted 8-bit Float to a Single-Precision Float

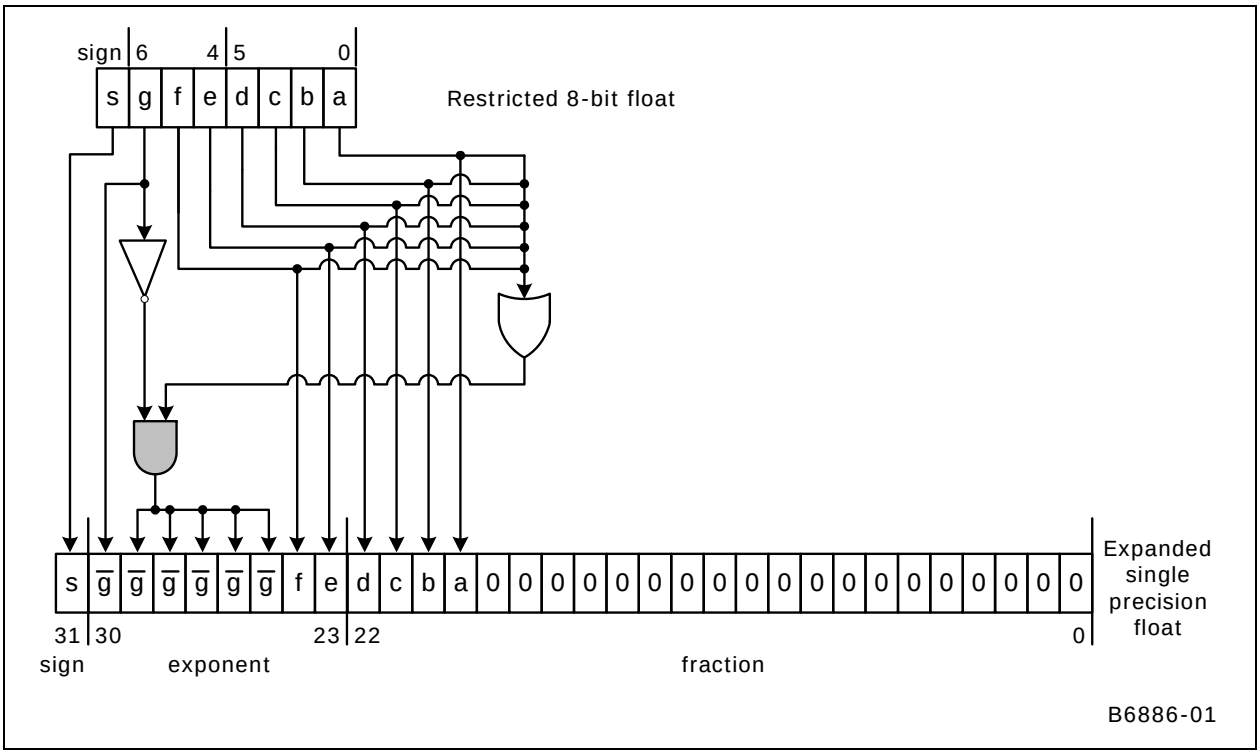


Table 4-2 shows all possible numbers of the restricted 8-bit float. Only normalized float numbers can be represented, including positive and negative zero, and positive and negative finite numbers. Normalized infinites, NaN and denormalized float numbers cannot be represented by this type. It should be noted that this 8-bit floating point format does not follow IEEE-754 convention in describing numbers with small magnitudes. Specifically, when the exponent field is zero and the fraction field is not zero, an implied one is still present instead of taking a denormalized form (without an implied one). This results in a simple implementation but with a smaller dynamic range – the magnitude of the smallest non-zero number is 0.125.

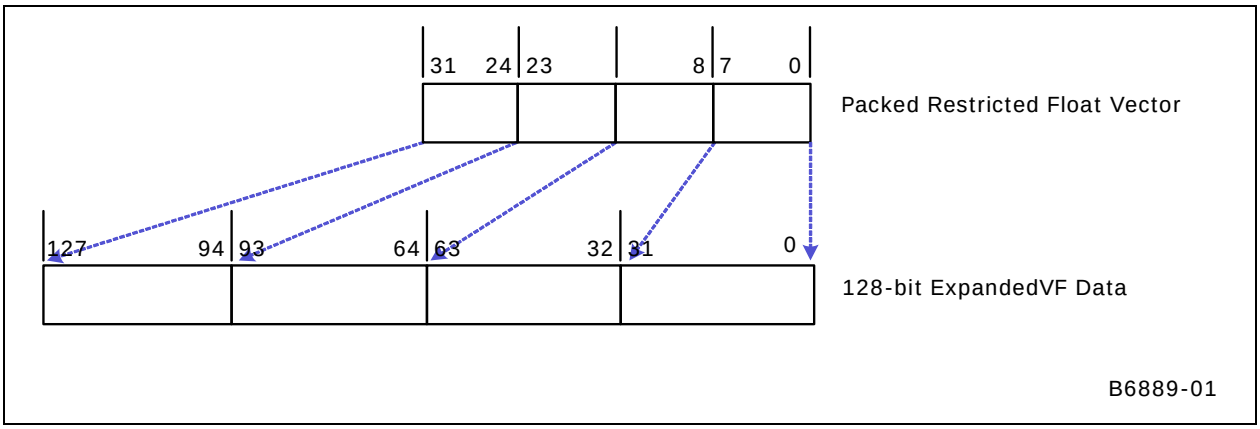
Table 4-2. Example of restricted 8-bit float numbers

Class	Restricted 8-bit Float				Extended 8-bit Exponent	Floating number in decimal
	Hex #	Sign [7]	Exponent [6:4]	Fraction [3:0]		
Positive Normalized Float	0x70-0x7F	0	**111**	0000 … 1111	**1**000 00**11**	16 … 31
	0x60-0x6F	0	**110**	0000 … 1111	**1**000 00**10**	8 … 15.5
	0x50-0x5F	0	**101**	0000 … 1111	**1**000 00**01**	4 … 7.75
	0x40-0x4F	0	**100**	0000 … 1111	**1**000 00**00**	2 … 3.875
	0x30-0x3F	0	**011**	0000 … 1111	**0**111 11**11**	1 … 1.9375
	0x20-0x2F	0	**010**	0000 … 1111	**0**111 11**10**	0.5 … 0.96875
	0x10-0x1F	0	**001**	0000 … 1111	**0**111 11**01**	0.25 … 0.484375
	0x01-0x0F	0	**000**	000**1** … 1111	**0**111 11**00**	0.125 … 0.2421875
	0x00	0	**000**	**0000**	0000 0000	0 (+zero)
Negative Normalized Float	0xF0-0xFF	1	**111**	0000 … 1111	**1**000 00**11**	-16 … -31
	0xE0-0xEF	1	**110**	0000 … 1111	**1**000 00**10**	-8 … -15.5
	0xD0-0xDF	1	**101**	0000 … 1111	**1**000 00**01**	-4 … -7.75
	0xC0-0xCF	1	**100**	0000 … 1111	**1**000 00**00**	-2 … -3.875
	0xB0-0xBF	1	**011**	0000 … 1111	**0**111 11**11**	-1 … -1.9375
	0xA0-0xAF	1	**010**	0000 … 1111	**0**111 11**10**	-0.5 … -0.96875
	0x90-0x9F	1	**001**	0000 … 1111	**0**111 11**01**	-0.25 … -0.484375
	0x81-0x8F	1	**000**	000**1** … 1111	**0**111 11**00**	-0.125 … -0.2421875
	0x80	1	**000**	**0000**	0000 0000	-0 (-zero)

Figure 4-6 shows the conversion of a packed exponent-only float to a 4-element vector of single precision floats.

The shorthand format notation for a packed signed half-byte vector is **VF**.

Figure 4-6. Converting a Packed Restricted Float Vector to a 128-bit Float Vector



4.3 Floating Point Modes

The architecture supports two floating point operation modes, namely IEEE floating point mode (IEEE mode) and alternative floating point mode (ALT mode). Both modes follow mostly the requirements in IEEE-754 but with different deviations. The deviations will be described in details in later sections. The primary difference between these modes is on the handling of Infs, NaNs and denorms. The IEEE floating point mode may be used to support newer versions of 3D graphics API Shaders and the alternative floating point mode may be used to support early Shader versions.

These two modes are supported by all units that perform floating point computations, including execution units, shared functions like Extended Math, the Sampler and the Render Cache color calculator, and fixed functions like VF, Clipper, SF and WIZ. Host software sets floating point mode through the fixed function state descriptors for 3D pipeline and the interface descriptor for media pipeline. Therefore different modes may be associated with different threads running concurrently. Floating point mode control for EU and shared functions are based on the floating point mode field (bit 0) of *cr0* register.

4.3.1 IEEE Floating Point Mode

4.3.1.1 Partial Listing of Honored IEEE-754 Rules

Here is a summary of expected 32-bit floating point behaviors in the architecture. Refer to IEEE-754 for topics not mentioned.

- INF – INF = NaN
- 0 * (+/–)INF = NaN
- 1 / (+INF) = +0 and 1 / (–INF) = –0
    - (+/–)INF / (+/–)INF = NaN as A/B = A * (1/B)
- INV (+0) = RSQ (+0) = +INF, INV (–0) = RSQ (–0) = –INF, and SQRT (–0) = –0
- RSQ (–finite) = SQRT (–finite) = NaN
- LOG (+0) = LOG (–0) = –INF, LOG (–finite) = LOG (–INF) = NaN

- NaN (any OP) any-value = NaN with one exception for min/max mentioned below. Resulting NaN may have different bit pattern than the source NaN.

- Normal comparison with conditional modifier of EQ, GT, GE, LT, LE, when either or both operands is NaN, returns FALSE. Normal comparison of NE, when either or both operands is NaN, returns TRUE.

    - Note: Normal comparison is either a **cmp** instruction or an instruction with conditional modifier

- Special comparison **cmpn** with conditional modifier of EQ, GT, GE, LT, LE, when the second source operand is NaN, returns TRUE, regardless of the first source operand, and when the second source operand is not NaN, but first one is, returns FALSE. **Cmpn** of NE, when the second source operand is NaN, returns FALSE, regardless of the first source operand, and when the second source operand is not NaN, but first one is, returns TRUE.

    - This is used to support the proposed IEEE-754R rule on **min** or **max** operations. For which, if only one operand is NaN, min and max operations return the other operand as the result.

- Both normal and special comparisons of any non-NaN value against +/– INF return exact result according to the conditional modifier. This is because that infinities are exact representation in the sense that +INF = +INF and –INF = –INF.

    - NaN is unordered in the sense that NaN != NaN.

- IEEE-754 requires floating point operations to produce a result that is the nearest representable value to an infinitely precise result, known as "round to nearest even" (RTNE). 32-bit floating point operations must produce a result that is within 0.5 Unit-Last-Place (0.5 ULP) of the infinitely precise result. This applies to addition, subtraction, and multiplication.

- All arithmetic floating point instructions does Round To Nearest Even at the end of the computation, except the round instructions.

4.3.1.2 Complete Listing of Deviations or Additional Requirements vs. IEEE-754

For a result that cannot be represented precisely by the floating point format, the execution unit uses rounding to nearest even to produce a result to an infinitely precise result. This ends up with a result that is within 0.5 Unit-Last-Place(0.5 ULP) of the infinitely precise result.

- **[DevSNB]** Denorms are flushed to sign-preserved zero on input and output of any floating point mathematical operation.

    - The exception to the above point about flushing denorms is that any I/O or data movement operation that does not manipulate the data (such as point sampling float data, or executing any raw “mov” instruction, or any sort of conditional raw “mov” if present) must not alter data at all (so a denorm remains denorm). Note that doing something that amounts to just moving data, but isn’t explicitly, such as multiplying a number by 1.0f is not detected as just a raw “mov”, and in this case a denorm flush would happen.

- NaN input to an operation obviously always produces NaN on output, however the exact bit pattern of the NaN is not required to stay the same (unless the operation is a raw “mov” instruction which does not alter data at all.)

- x*1.0f must always result in x (except denorm flushed and possible bit pattern change for NaN).

- x +/- 0.0f must always result in x (except denorm flushed and possible bit pattern change for NaN). But -0 + 0 = +0.

- Fused operations (such as mac, dp4, dp3, etc.) may produce intermediate results out of 32-bit float range, but whose final results would be within 32-bit float range if intermediate results were kept at greater precision. In this case, implementations are permitted to produce either the correct result, or else +/-INF. Thus, compatibility between a fused operation, such as “mac”, with the unfused equivalent, “mul” followed by “add” in this case, is not guaranteed.

    - As the accumulator registers have more precision than 32-bit float, any instruction with accumulator as a source/destination operand may produce a different result than that using GRF/MRF registers.

- API Shader divide operations are implemented as x*(1.0f/y). With the two-step method, x*(1.0f/y), the multiply and the divide each independently operate at the 32-bit floating point precision level (accuracy to 1 ULP).

- See the Type Conversion section below for rules on converting to/from float representations.

4.3.1.3 Comparison of Floating Point Numbers

The following tables (Table 4-3 through Table 4-8) detail the rules for floating point comparison. In the tables, "+/-Fin" stands for a positive or negative finite precision floating point number. Result is either a true (T) or false (FALSE or F). Each row corresponds to a fixed <src0> and each column corresponds to a fixed <src1>. When comparing two positive finite numbers (or two negative finite numbers), the result can be T or F depending on the values. Therefore, the corresponding fields in the following tables are marked as T/F.

Table 4-3. Results of "Greater-Than" Comparison – CMP.G

src0 src1	-inf	-Fin	-denorm	-0	+0	+denorm	+Fin	+inf	NaN
-inf	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
-Fin	T	T/F	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
-denorm	T	T	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
-0	T	T	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
+0	T	T	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
+denorm	T	T	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
+Fin	T	T	T	T	T	T	T/F	FALSE	FALSE
+inf	T	T	T	T	T	T	T	FALSE	FALSE
NaN	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

Table 4-4. Results of "Less-Than" Comparison – CMP.L

src0 src1	-inf	-Fin	-denorm	-0	+0	+denorm	+Fin	+inf	NaN
-inf	FALSE	T	T	T	T	T	T	T	FALSE
-Fin	FALSE	T/F	T	T	T	T	T	T	FALSE
-denorm	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	T	T	FALSE
-0	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	T	T	FALSE
+0	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	T	T	FALSE
+denorm	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	T	T	FALSE
+Fin	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	T/F	T	FALSE
+inf	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
NaN	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

Table 4-5. Results of "Equal-To" Comparison – CMP.E

src0 src1	-inf	-Fin	-denorm	-0	+0	+denorm	+Fin	+inf	NaN
-inf	T	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
-Fin	FALSE	T/F	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
-denorm	FALSE	FALSE	T	T	T	T	FALSE	FALSE	FALSE
-0	FALSE	FALSE	T	T	T	T	FALSE	FALSE	FALSE
+0	FALSE	FALSE	T	T	T	T	FALSE	FALSE	FALSE
+denorm	FALSE	FALSE	T	T	T	T	FALSE	FALSE	FALSE
+Fin	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	T/F	FALSE	FALSE
+inf	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	T	FALSE
NaN	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

Table 4-6. Results of "Not-Equal-To" Comparison – CMP.NE

src0 src1	-inf	-Fin	-denorm	-0	+0	+denorm	+Fin	+inf	NaN
-inf	FALSE	T	T	T	T	T	T	T	T
-Fin	T	T/F	T	T	T	T	T	T	T
-denorm	T	T	FALSE	FALSE	FALSE	FALSE	T	T	T
-0	T	T	FALSE	FALSE	FALSE	FALSE	T	T	T
+0	T	T	FALSE	FALSE	FALSE	FALSE	T	T	T
+denorm	T	T	FALSE	FALSE	FALSE	FALSE	T	T	T
+Fin	T	T	T	T	T	T	T/F	T	T
+inf	T	T	T	T	T	T	T	FALSE	T
NaN	T	T	T	T	T	T	T	T	T

Table 4-7. Results of "Less-Than Or Equal-To" Comparison – CMP.LE

src0 src1	-inf	-Fin	-denorm	-0	+0	+denorm	+Fin	+inf	NaN
-inf	T	T	T	T	T	T	T	T	FALSE
-Fin	FALSE	T/F	T	T	T	T	T	T	FALSE
-denorm	FALSE	FALSE	T	T	T	T	T	T	FALSE
-0	FALSE	FALSE	T	T	T	T	T	T	FALSE
+0	FALSE	FALSE	T	T	T	T	T	T	FALSE
+denorm	FALSE	FALSE	T	T	T	T	T	T	FALSE
+Fin	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	T/F	T	FALSE
+inf	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	T	FALSE
NaN	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

Table 4-8. Results of "Greater-Than or Equal-To" Comparison – CMP.GE

src0 src1	-inf	-Fin	-denorm	-0	+0	+denorm	+Fin	+inf	NaN
-inf	T	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
-Fin	T	T/F	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
-denorm	T	T	T	T	T	T	FALSE	FALSE	FALSE
-0	T	T	T	T	T	T	FALSE	FALSE	FALSE
+0	T	T	T	T	T	T	FALSE	FALSE	FALSE
+denorm	T	T	T	T	T	T	FALSE	FALSE	FALSE
+Fin	T	T	T	T	T	T	T/F	FALSE	FALSE
+inf	T	T	T	T	T	T	T	T	FALSE
NaN	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

4.3.1.4 Min/Max of Floating Point Numbers

A special comparison called Compare-NaN is introduced to handle the difference of above mentioned floating point comparison and the rules on supporting MIN/MAX. To compute the MIN or MAX of two floating point numbers, if one of the numbers is NaN and the other one is not, MIN or MAX of the two numbers returns the one that is not NaN. When two numbers are NaN, MIN or MAX of the two numbers always returns source1.

Mix and Max is achieved using conditional selects, i.e., SEL with 'condition modifiers' as :

Evaluations	Instructions
MIN(src0, src1) = (src0 < src1) ? src0 : src1	sel.l.f0.0 dst src0 src1
MAX(src0, src1) = (src0 >= src1) ? src0 : src1	sel.ge.f0.0 dst src0 src1

Note even f0.0 is specified in the instruction, the flag register is not touched by this instruction.

The following tables (**Error! Reference source not found.** through Table 4-10) detail the rules for this special compare-NaN operation for floating point numbers. Notice that excepting "Not-Equal-To" comparison-NaN, last columns in all other tables have 'T'.

Table 4-9. Results of "Less-Than" Comparison-NaN – CMPN.L

src0 src1	-inf	-Fin	-denorm	-0	+0	+denorm	+Fin	+inf	NaN
-inf	FALSE	T	T	T	T	T	T	T	T
-Fin	FALSE	T/F	T	T	T	T	T	T	T
-denorm	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	T	T	T
-0	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	T	T	T
+0	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	T	T	T
+denorm	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	T	T	T
+Fin	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	T/F	T	T
+inf	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	T
NaN	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

Table 4-10. Results of "Greater-Than or Equal-To" Comparison-NaN – CMPN.

src0 src1	-inf	-Fin	-denorm	-0	+0	+denorm	+Fin	+inf	NaN
-inf	T	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	T
-Fin	T	T/F	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	T
-denorm	T	T	T	T	T	T	FALSE	FALSE	T
-0	T	T	T	T	T	T	FALSE	FALSE	T
+0	T	T	T	T	T	T	FALSE	FALSE	T
+denorm	T	T	T	T	T	T	FALSE	FALSE	T
+Fin	T	T	T	T	T	T	T/F	FALSE	T
+inf	T	T	T	T	T	T	T	T	T
NaN	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

Table 4-11. Results of "Greater-Than or Equal-To" Comparison-NaN – CMPN.GE

src0 src1	-inf	-Fin	-denorm	-0	+0	+denorm	+Fin	+inf	NaN
-inf	T	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	T
-Fin	T	T/F	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	T
-denorm	T	T	T	T	T	T	FALSE	FALSE	T
-0	T	T	T	T	FALSE	T	FALSE	FALSE	T
+0	T	T	T	T	T	T	FALSE	FALSE	T
+denorm	T	T	T	T	T	T	FALSE	FALSE	T
+Fin	T	T	T	T	T	T	T/F	FALSE	T
+inf	T	T	T	T	T	T	T	T	T
NaN	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

4.3.2 Alternative Floating Point Mode

The key characteristics of the alternative floating point mode is that NaN, Inf and denorm are not expected for an application to pass into the graphics pipeline, and the graphics hardware must not generate NaN, Inf or denorm as computation result. For example, a result that is larger than the maximum representable floating point number is expected to be flushed to the largest representable floating point number, i.e., +FLT_MAX. The FLT_MAX has an exponent of 0xFE and a mantissa of all one's, which is the same for IEEE floating point mode.

Here is the complete list of the differences of legacy graphics mode from the relaxed IEEE-754 floating point mode.

- Any +/- INF result must be flushed to +/- FLT_MAX, instead of being output as +/- INF.
- Extended mathematics functions of log(), rsq() and sqrt() take the absolute value of the sources before computation to avoid generating INF and NaN results.

Table 4-12 shows the support of these differences in various hardware units.

Table 4-12. Supported Legacy Float Mode and Impacted Units

IEEE-754 Deviations	VF	Clipper	SF	WIZ	EU	EM	Sampler	RC
Any +/- INF result flushed to +/- FLT_MAX	Y	Y	Y	Y	Y	Y	Y	Y
Log, rsq, sqrt take abs() of sources	N/A	N/A	N/A	N/A	N/A	Y	N/A	N/A

Table 4-13 shows some of the desired or recommended alternative floating point mode behaviors that do not have hardware design impact. The reasons of not needing special hardware support for these items are also provided.

Table 4-13. Dismissed legacy behaviors

Suggested IEEE-754 Deviations	Reason for Dismiss
Mov forces (+/-)INF to (+/-)FLT_MAX	(+/-)INF is never present as input
(+/-)INF – (+/-)INF = +/- FLT_MAX instead of NaN	(+/-)INF is never present as input
Denorm must be flushed to zero in all cases (including trivial mov and point sampling)	Denorm is never present as input
Anything*0=0 (including NaN*0=0 and INF*0=0)	NaN and INF are never present as input
Except propagated NaN, NaN is never generated	NaN is never present as input and he system never generates NaN based on rules in the previous table
An input NaN gets propagated excepting (a)-(d)	NaN is never present as input
(a) Rcp (and rsq) of 0 yields FLT_MAX	N/A, as it is already covered by the general rule “Any +/- INF result flushed to +/- FLT_MAX”
(b) Sampler honors 0/0 = 0 as if (1/0)*0	There is no divide in Sampler
I Rcp (and rsq) of INF yields +/- 0	(+/-)INF is never present as input
(d) Sampler honors INF/INF = 0 as if (1/INF)=0 followed by Anything*0 = 0	There is no divide in Sampler

4.4 Type Conversion

4.4.1 Float to Integer

Converting from float to integer is based on rounding toward zero. If the floating point value is +0, -0, +Denorm, -Denorm, +NaN –r -NaN, the resulting integer value is always 0. If the floating point value is positive infinity (or negative infinity), the conversion result takes the largest (or the smallest) represent-able integer value. If the floating point value is larger (or smaller) than the largest (or the smallest) represent-able integer value, the conversion result takes the largest (or the smallest) represent-able integer value. The following table shows these special cases. The last two rows are just examples. They can be any number outside the represent-able range of the output integer type (UD, D, UW, W, UB and B).

Input Format	Output Format					
F	UD	D	UW	W	UB	B
+/- Zero	00000000	00000000	00000000	00000000	00000000	00000000
+/- Denorm	00000000	00000000	00000000	00000000	00000000	00000000
NAN	00000000	00000000	00000000	00000000	00000000	00000000
-NAN	00000000	00000000	00000000	00000000	00000000	00000000
INF	FFFFFFFF	7FFFFFFF	0000FFFF	00007FFF	000000FF	0000007F

Input Format	Output Format					
-INF	00000000	80000000	00000000	00008000	00000000	00000080
$+2^{32}$ (*)	FFFFFFFF	7FFFFFFF	0000FFFF	00007FFF	000000FF	0000007F
$-2^{32}-1$ (*)	00000000	80000000	00000000	00008000	00000000	00000080

4.4.2 Integer to Integer with Same or Higher Precision

Converting an unsigned integer to a signed or an unsigned integer with higher precision is based on zero extension.

Converting an unsigned integer to a signed integer with the same precision is based on modular wrap-around. Without saturation, a larger than represent-able number becomes a negative number. With saturation, a larger than represent-able number is saturated to the largest positive represent-able number.

Converting a signed integer to a signed integer with higher precision is based on sign extension.

Converting a signed integer to an unsigned integer with higher precision is based on zero extension. Without saturation, a negative number becomes a large positive number with the sign bit wrapped-up. With saturation, a negative number is saturated to zero.

4.4.3 Integer to Integer with Lower Precision

Converting a signed or an unsigned integer to a signed or an unsigned integer with lower precision is based on bit truncation. Without saturation, only the lower bits are kept in the output regardless of the sign-ness of input and output. With saturation, a number that is outside the represent-able range is saturated to the closest represent-able value.

4.4.4 Integer to Float

Converting a signed or an unsigned integer to a single precision float number is to round to the closest representable float number. For any integer number with magnitude less than or equal to 24 bits, resulting float number is a precise representation of the input. However, if it is more than 24 bits, by default a “round to nearest even” is performed.

5. Execution Environment

5.1 Overview

The instruction set is a general-purpose data-parallel instruction set optimized for graphics and media computations. Supports for 3D graphics API (application program interface) Shader instructions are mostly native, meaning that DevSNB provides efficient execution for Shader programs. Depending on the Shader program operation modes (for example, a Vertex Shader may be executed on a base of a vertex-pair, while a Pixel Shader may be executed on a base of a 16-pixel group), translation from 3D graphics API Shader instruction streams into native instructions may be required. In addition, there are many specific capabilities to accelerate media applications. The following list provides a summary of the instruction set.

- ISA support SIMD (single instruction multiple data) instructions. The number of data elements per instruction depends on the data type.
- ISA supports SIMD parallel arithmetic, vector arithmetic, logical, and SIMD control/branch instructions.
- ISA supports instruction level variable-width SIMD execution.
- ISA supports conditional SIMD execution via destination mask, predication, and execution mask.
- ISA supports instruction compaction.
- An instruction may be executed in multiple cycles over a SIMD execution pipeline.
- Most instructions have three operands. Some instructions have additional implied source and destination operands. Some instructions have explicit dual destinations.
- ISA supports region-based register addressing.
- ISA supports direct and indirect (indexed) register addressing.
- instructions may have a scalar and vector immediate source operand.
- Higher precision accumulator registers are architecturally visible.
- Self-modifying code is not allowed (instruction streams, including instruction caches, are read-only).

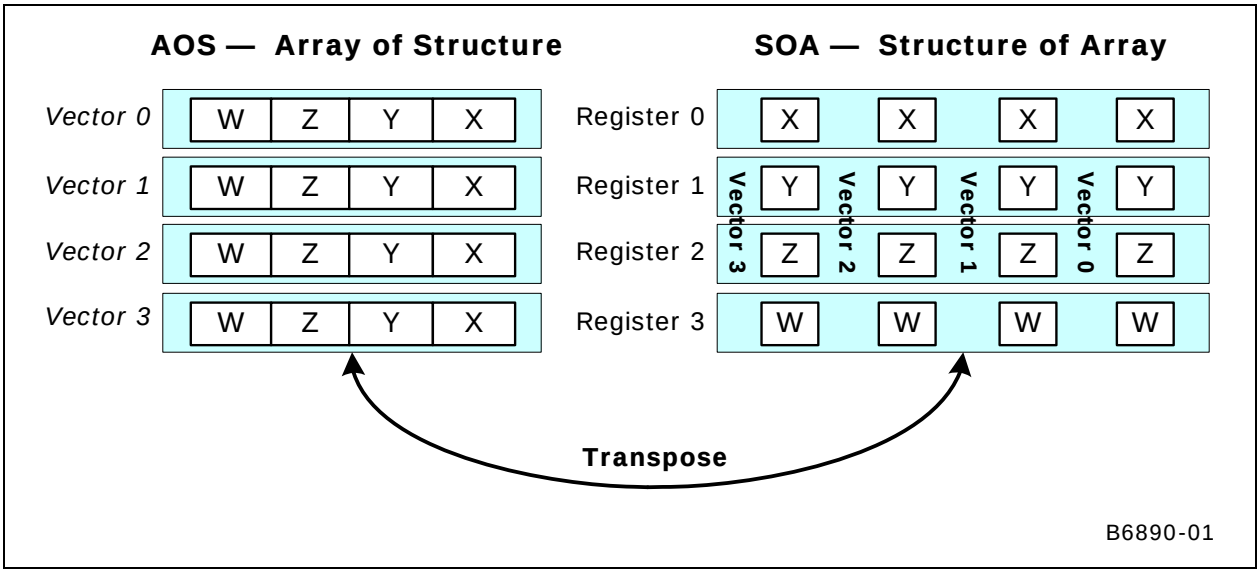
5.2 Primary Usage Models

In describing the usage models of the instruction set, it is inevitable to forward reference terminology, syntax and instructions detailed later in this specification. For clarity reasons, not all forward references will be provided in this section as well as subsequent sections. For example, reference to binary instruction fields such as ***Align1, Align16, Compr, SecHalf***, etc, can be found in the Instruction Summary chapter. And assembly instruction syntax can be found in the Instruction Summary chapter and Instruction Reference chapter.

5.2.1 AOS and SOA Data Structures

With the Align1 and Align16 access modes, the instruction set provides effective SIMD computation regardless whether data are arranged in array of structure (AOS) form or in structure of array (SOA) form. The AOS and SOA data structures are illustrated by the examples in Figure 5-1. The example shows two different ways of storing four vectors in four SIMD registers. For simplicity, data vector and SIMD register both have four data elements. The four data elements in a vector are denoted by X, Y, Z and W just as for a vertex in 3D geometry. The AOS structure stores one vector in a register and the next vector in another register. The SOA structure stores one data element of each vector in a register and the next element of each vector in the next register and so on. It is obvious in this case the two structures can be related by a matrix transpose operation.

Figure 5-1. AOS and SOA data structures



3D and media applications take advantage of such broad architecture support and use both AOS and SOA data arrangements.

- Vertices in 3D Geometry (Vertex Shader and Geometry Shader) are arranged in AOS structure and run on SIMD4x2 and SIMD4 modes, respectively, as detailed below.
- Pixels in 3D Rasterization (Pixel Shader) are arranged in SOA structure and run on SIMD8 and SIMD16 modes as detailed below.

- Pixels in media are primarily arranged in SOA structure, and occasionally in AOS structure with possible mixed mode of operations that use region-based addressing extensively.

These are preferred methods; alternative arrangements may also be possible. Shared function resources provide data transpose capability to support both modes of operations: The sampler has a transpose for sample reads, the data port has a transpose for render cache writes, and the URB unit has a transpose for URB writes.

The following 3D graphics API Shader instruction will be used in the following sections to illustrate various modes of operations:

add <dst>.xyz <src0>.yxzw <src1>.zwxy

This example is an SIMD instruction that takes two source operands <src0> and <src1>, performs addition operation (add), and store the additions to the destination operand <dst>. Each operand contains four floating point data elements. The data type is determined by the instruction opcode. This instruction also uses source swizzle modifier (.yxzw for <src0> and .zwxy for <src1> and destination mask modifier (.xyz). Please refer to programming specifications of 3D graphics API Shader instructions for more details.

A physical GRF register has 256 bits, which may be used to store 8 floating point data elements. For 3D graphics usage, the mode of operation is (loosely) termed after the data structure as SIMD*mxn*, where "*m*" is a numerical term describing the size of vector and "*n*" is the number of concurrent program flows executed in SIMD.

- Execution with AOS data structures
  - **SIMD4 (short for SIMD4x1)** stands for the mode of operation where a SIMD instruction operates on 4-element vectors stored packed in the registers. There is only one program flow.
  - **SIMD4x2** standards for the SIMD operation based on a pair of 4-element vectors stored in a register. There are effectively two programs running side by side with one vector per program.
- Execution with SOA data structures – also referred to as "channel serial" execution
  - **SIMD8 (short for SIMD1x8)** standards for the SIMD operation based on the SOA data structure where one register contains one data element (the same one) of 8 vectors. Effectively, there are 8 concurrent program flows.
  - **SIMD16 (short for SIMD1x16)** is a special term indicating the use of instruction compression whereas each compressed SIMD instruction operates on a pair of registers that contains one data element (the same one) of 16 vectors. SIMD16 has 16 concurrent program flows.

5.2.2 SIMD4 Mode of Operation

With a register mapping of <src0> to doublewords 0-3 of *r2*, <src1> to doublewords 4-7 of *r2* and <dst> to doublewords 0-3 of *r3*, the example 3D graphics API Shader instruction can be translated into the following instruction:

add (4) r3<4>.xyz:f r2<4>.yzwx:f r2.4<4>.zwxy:f {NoMask}

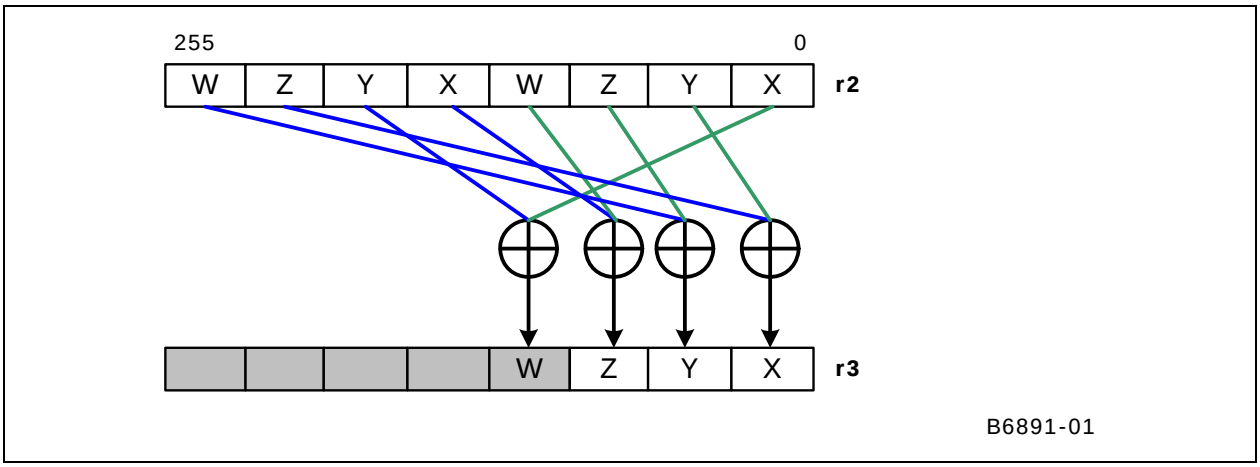
Without diving too much into the syntax definition of an instruction, it is clear that an instruction also takes two source operands and one destination operands. The second term, (4), is the execution size that determines the number of data elements processed by the SIMD instruction. It is similar to the term SIMD Width used in the literature. Each operand is described by the register region parameters such as '<4>' and data type (e.g. "*:f*"). These will be detailed in Section 5.3. The instruction option field, {NoMask}, ensure that the execution occurs for the execution channels shown in the instruction, instead of, possibly, being masked out by the conditional masks of the thread (See Instruction Summary chapter for definition of *MaskCtrl* instruction field).

The operation of this instruction is illustrated in Figure 5-2. In this example, both source operands share the same physical GRF register r2. The two are distinguished by the subregister number. The source swizzles control the routing of source data elements to the parallel adders corresponding to the destination data elements. The shaded areas in the destination register r3 are not modified. In particular, doublewords 4-7 are unchanged as the execution size is 4; doubleword 3 is unchanged due to the destination mask setting.

In this mode of operation, there is only one program flow – any branch decision will be based on a scalar condition and apply to the whole vector of four elements. Option {*NoMask*} ensures that the instruction is not subject to the masks. In fact, most of the instructions in a thread should have {*NoMask*} set.

Even though the execution only performs four parallel add operations, the GEN instruction still executes in 2 cycles (with no useful computation in the second cycle).

Figure 5-2. A SIMD4 Example



5.2.3 SIMD4x2 Mode of Operation

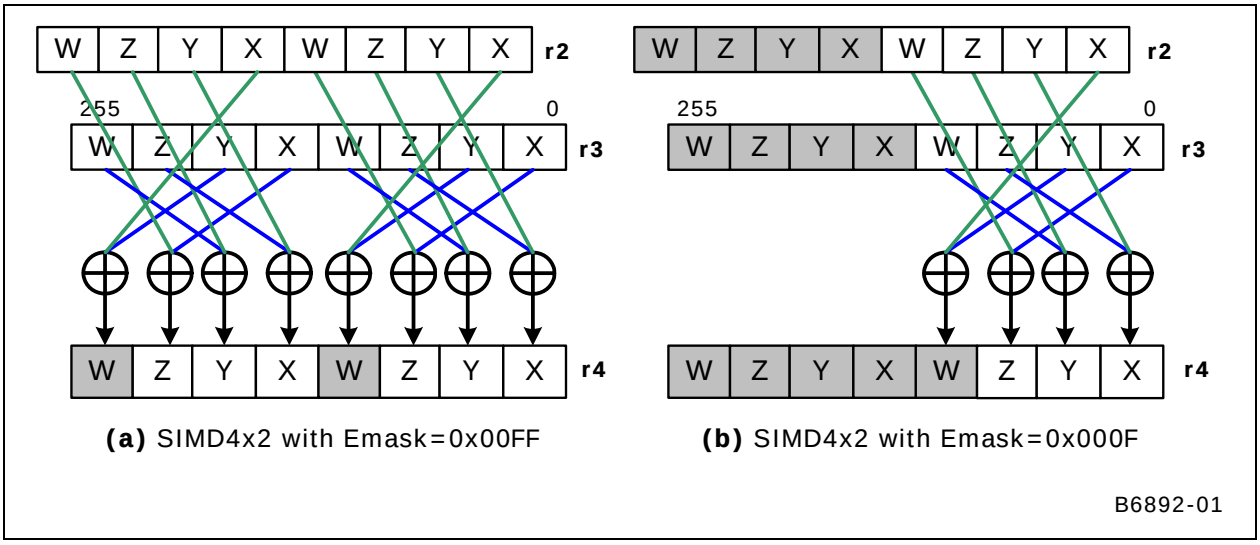
In this mode, two corresponding vectors from the two program flows fill a physical register. With a register mapping of <src0> to r2, <src1> to r3 and <dst> to r4, the example 3D graphics API Shader instruction can be translated into the following instruction:

add (8) r4<4>.xyz:f r2<4>.yxzw:f r3<4>.zwxy:f

This instruction is subject to the execution mask, which initiated from the dispatch mask. If both program flows are available (e.g. Vertex Shader executed with two active vertices), the dispatch mask is set to 0x00FF. The operation of this instruction is illustrated in Figure 5-3 (a). The source swizzles control the routing of source data elements to the parallel adders corresponding to the destination data elements. The shaded areas in the destination register r3 (doublewords 3 and 7) are unchanged due to the

destination mask setting. If only one program flow is available (e.g. the same SIMD4x2 Vertex Shader with only one active vertex), the dispatch mask is set to 0x000F. The operation of the same instruction is shown in Figure 5-3(b).

Figure 5-3. SIMD4x2 Examples with Different Emasks

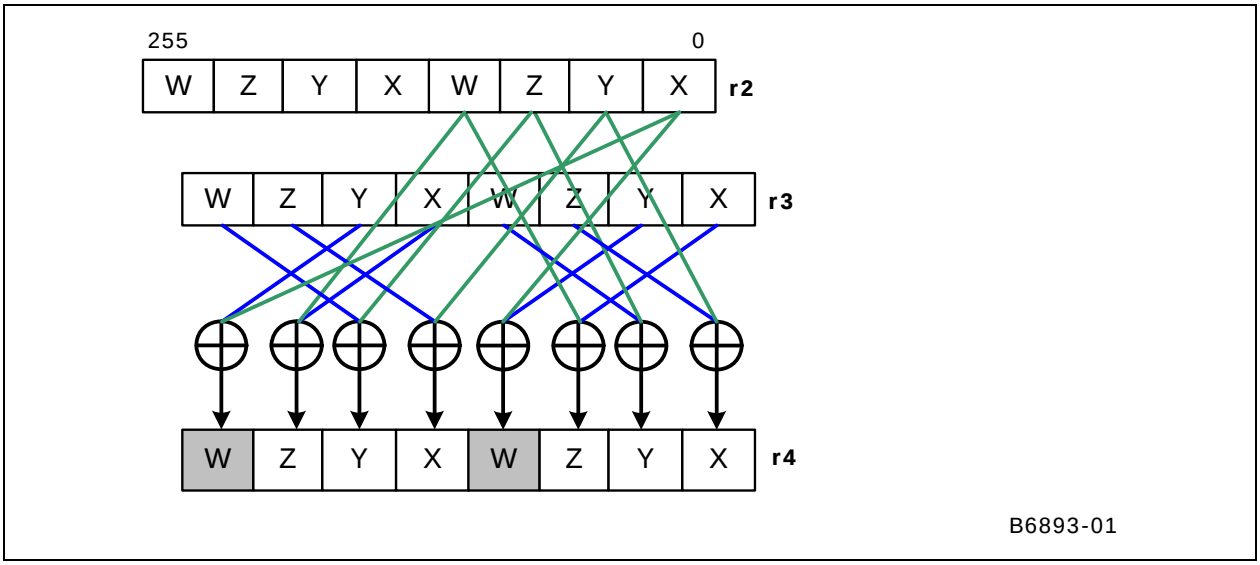


The two source operands only need to be 16-byte aligned, not have to be GRF register aligned. For example, the first source operand could be a 4-element vector (e.g. a constant) stored in doublewords 0-3 in r2, which is shared by the two program flows. The example 3D graphics API Shader instruction can then be translated into the following instruction:

*add (8) r4<4>.xyz:f r2<**0**>.yzwx:f r3<4>.zwxy:f*

The only difference here is that the vertical stride of the first source is 0. The operation of this instruction is illustrated in Figure 5-4.

Figure 5-4. A SIMD4x2 Example with a Constant Vector Shared by Two Program Flows



5.2.4 SIMD16 Mode of Operation

With 16 concurrent program flows, one element of a vector would take two GRF registers. In this mode, two corresponding vectors from the two program flows fill a physical register.

With the following register mappings,

<src0>: *r2-r9 (with 16 X data elements in r2-r3, Y in r4-5, Z in r6-7 and W in r8-9),*

<src1>: *r10-r17,*

<dst>: *r18-r25,*

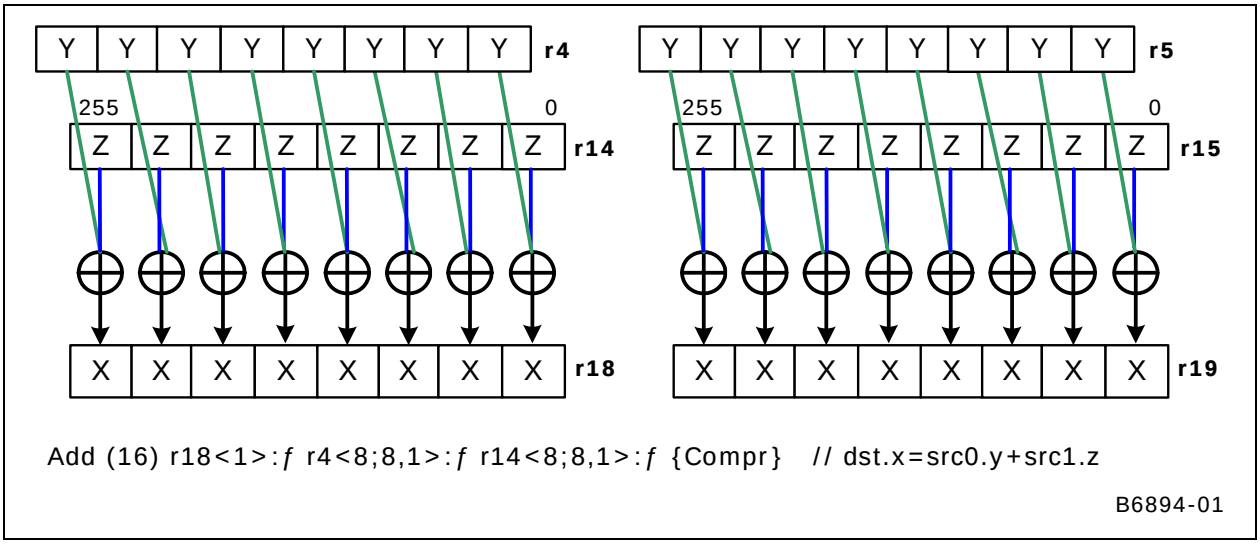
the example 3D graphics API Shader instruction can be translated into the following three instructions:

```
add (16)  r18<1>:f  r4<8;8,1>:f  r14<8;8,1>:f     // dst.x = src0.y + src1.z
add (16)  r20<1>:f  r6<8;8,1>:f  r16<8;8,1>:f     // dst.y = src0.z + src1.w
add (16)  r22<1>:f  r8<8;8,1>:f  r10<8;8,1>:f     // dst.z = src0.w + src1.x
```

The three instructions correspond to the three enabled destination masks As there is no output for the W elements of <dst>, no instruction is needed for that element. The first instruction inputs the Y elements of <src0> and the Z elements of <src1> and outputs the X elements of <dst>. The operation of this instruction is shown in Figure 5-5.

With the number of program flows more than one, the above instructions also subject to execution mask. The 16-bit dispatch mask is partitioned into four groups with four bits each. For Pixel Shader generated by the Windower, each 4-bit group corresponds to a 2x2 pixel subspan. If a subspan is not valid for a Pixel Shader instance, the corresponding 4-bit group in the dispatch mask is not set. Therefore, the same instructions can be used independent of the number of available subspans without creating bogus data in the subspans that are not valid.

Figure 5-5. A SIMD16 Example



Similar to SIMD4x2 mode, a constant may also be shared for the 16 program flows. For example, the first source operand could be a 4-element vector (e.g. a constant) stored in doublewords 0-3 in r2 (AOS format). The example 3D graphics API Shader instruction can then be translated into the following instruction:

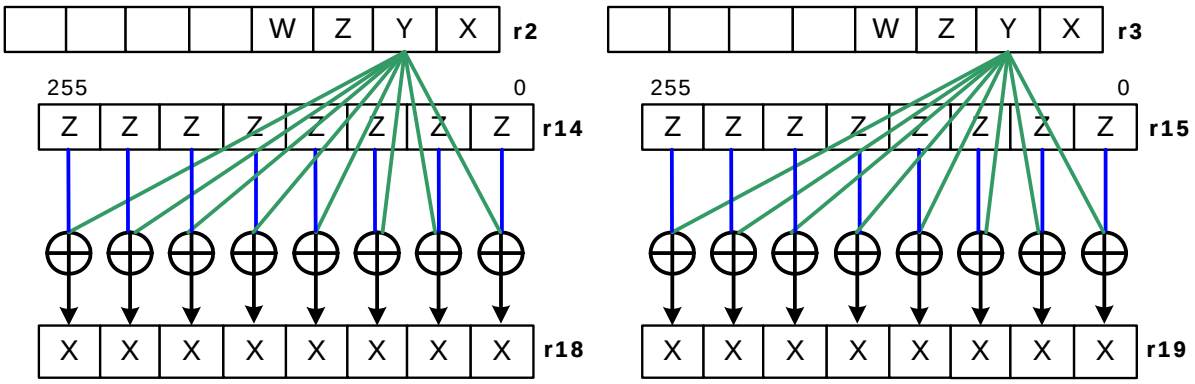
add (16) r18<1>:f r2.1<0;1,0>:f r14<8;8,1>:f {Compr} // dst.x = src0.y + src1.z

add (16) r20<1>:f r2.2<0;1,0>:f r16<8;8,1>:f {Compr} // dst.y = src0.z + src1.w

add (16) r22<1>:f r2.3<0;1,0>:f r10<8;8,1>:f {Compr} // dst.z = src0.w + src1.x

The register region of the first source operand represents a replicated scalar. The operation of the first instruction is illustrated in Figure 5-6.

Figure 5-6. Another SIMD16 Example with an AOS Shared Constant



Add (16) r18<1>:f r2.1<0;1,0>:f r14<8;8,1>:f {Compr} // dst.x=src0.y+src1.z

B6895-01

5.2.5 SIMD8 Mode of Operation

Each compressed instruction has two correspond uncompressed instructions. Taking the example instruction shown in Figure 5-6, it is equivalent to the following two instructions.

add (8) r18<1>:f r4<8;8,1>:f r14<8;8,1>:f // dst.x[7:0] = src0.y + src1.z

add (8) r19<1>:f r5<8;8,1>:f r15<8;8,1>:f {SecHalf} // dst.x[15:8] = src0.y + src1.z

Therefore, SIMD8 can be viewed as a special case for SIMD16.

There are other reasons that SIMD8 instructions may be used. Within a program with 16 concurrent program flows, some time SIMD8 instruction must be used due to architecture restrictions. For example, the address register a0 only have 8 elements, if an indirect GRF addressing is used, SIMD16 instructions are not allowed.

5.3 Registers and Register Regions

5.3.1 Register Files

Registers are grouped into different name spaces called register files. There are three different register files defined: General Register File, Message Register File, and Architecture Register File. In addition, immediate operands also have a unique encoding of the register file field, even though they come inline in the instruction word and do not have dedicated physical storages.

- General Register File (GRF): GRF contains general-purpose read-write registers.
- Message Register File (MRF): MRF contains special purpose registers used for message passing only. MRF registers are write-only.
- Architecture Register File (ARF): ARF contains all other architectural registers, including the address registers (*a#*), accumulators (*acc#*), flags (*f#*), masks (*mask#*), mask stack (*ms#*), mask stack depth (*msd#*), notification count (*n#*), instruction pointer (*ip*), etc. Null register (*null*) is also encoded as an ARF register.
- Immediate: Certain instructions take immediate terms as the source operands. These immediate terms have a distinct register file encoding.

Each thread executed in an EU has its own thread context, i.e. dedicated register space, which is not shared between threads executing on a common EU or on a different EU. In the rest of the Chapters, register access are in respect to a given thread.

5.3.2 GRF Registers

Number of Registers:	Various
Default Value:	None
Normal Access:	RW
Elements:	Various
Element Size:	Various
Element Type:	Various
Access Granularity:	Byte
Write Mask Granularity:	Byte
Index-ability:	Yes

Registers in the General Register File are the most commonly used read-write registers. During the execution of a thread, GRF registers are used to store the temporary data, and serve as the destination to receive data from shared function units (and some times from a fixed function unit). They are also used to store the input (initialization) data when a thread is created. By allowing fixed function hardware to initialize some portion of GRF registers during thread dispatch time, the architecture can achieve better parallelism. A thread's execution efficiency can also be improved as some data are already in the register to be executed upon. Besides these registers containing thread's payload, the rest of GRF registers of a thread are not initialized.

Table 5-1. Summary of GRF Registers

Register File	Register Name	Description
General Register File (GRF)	***r#***	General purpose read write registers

Each execution unit has a fixed size physical GRF register RAM. The GRF register RAM is shared by all threads on the EU. GRF space for a thread is allocated at thread dispatch time, allowing the amount of GRF space to adapt to the need of a given thread.

Mapping of a thread's GRF registers to the physical GRF RAM is through a translation table. Therefore, a thread's access to GRF is always through the 0-based logical view. For example, the GRF registers are *r0* through *r127*.

GRF registers can be accessed using region-based addressing at byte granularity (both read and write). A source operand must be contained within two adjacent physical registers. A destination operand must be contained within one physical register. GRF registers support direct addressing and register-indirect addressing. Register-indirect addressing uses the address registers (ARF registers a#) and an immediate address offset value.

When accessing (read and/or write) outside the GRF register range allocated for a given thread either through direct or indirect addressing, the result is unpredictable.

Table 5-2. GRF Registers Available in Device Hardware

Device	Physical Register Size	Allocation Granularity	Number per Thread	Number per EU
[DevSNB]	256 bits	Fixed allocation of 128 register	128 registers	640 registers

5.3.3 MRF Registers

Number of Registers:	Fixed
Default Value:	None
Normal Access:	WO
Elements:	Various
Element Size:	Various
Element Type:	Various
Access Granularity:	Byte
Write Mask Granularity:	Byte
Index-ability:	Yes

Registers in the Message Register File are used to store the header and payload for out-going messages from a thread to a shared function such as the Sampler. There are fixed number of MRF registers for each thread.

MRF registers are write-only, and therefore, can only be the destination operand of an instruction.

MRF registers support write-enable at byte granularity. When an MRF register is used as the current destination operand of the *send* instruction, only 256-bit register aligned access is supported.

When accessing (write) outside the MRF register range for a given thread, the result is unpredictable.

Table 5-3. Summary of MRF Registers

Register File	Register Name	Description
Message Register File (MRF)	***m#***	Special purpose output write-only registers

Table 5-4. MRF Registers Available in Device Hardware

Device	Physical Register Size	Number per Thread	Indirect Addressing?
[DevSNB]	256 bits	24 registers	Yes

5.3.4 ARF Registers

5.3.4.1 Overview

Besides GRF and MRF registers that are directly indicated by unique register file coding, all other registers belong to the general Architecture Register File (ARF). Encoding of architecture register types are based on the MSBs of the register number field, RegNum, in the instruction word. RegNum field has 8 bits. The 4 MSBs, RegNum[7:4], represent the architecture register type.

Table 5-5. Summary of Architecture Registers [DevSNB+]

Register Type (RegNum [7:4])	Register Name	Register Count	Description
0000	***null***	*1*	Null register
0001	***a0.#***	*1*	Address register
0010	***acc#***	***2***	Accumulator register
0011	***f0.#***	*1*	Flag register **[DevSNB]**
0011	***reserved***		
0100	***reserved***		
0110	***reserved***		
0111	***sr0.#***	*1*	State register
1000	***cr0.#***	*1*	Control register
1001	***n#***	*2*	Notification count register
1010	***ip***	*1*	Instruction pointer register
1011	***tdr***	*1*	Thread dependency register
1100	***reserved***		
1101	*reserved*		
1111	***reserved***		

The remaining register number field RegNum[3:0] is used to identify the register number of a given architecture register type. Therefore, maximum number of registers for a given architecture register type is limited to 16. The subregister number field, SubRegNum, in instruction word has 5 bits. It is used for addressing subregister region for an architecture register supporting register subdivision. SubRegNum field is in unit of byte. Therefore, maximum number of bytes of an architecture register is limited to 32. Depending on alignment restriction of a register type, only certain encodings of SubRegNum field is applicable for an architecture register. The detailed definitions are provided in the following sections.

5.3.4.2 Access Granularity

ARF registers may be accessed with subregister granularity according to the descriptions below and following the same rule of region-based addressing for GRF and MRF. The machine code for register number and subregister number of ARF follows the same rule as for other register files with byte granularity. For an ARF as a source operand, the region-based address controls the source swizzle mux. The destination subregister number and destination horizontal stride can be used to control to generate the destination write mask at byte level.

A special restriction on region-based addressing for ARF is that the register region cannot cross register boundary. This rule in fact only applies to the accumulator as it is the only ARF register containing multiple registers (two).

Subregister fields of an ARF register may not all populated (indicated by the subregister indicated as reserved). Write to an unpopulated subregister will be dropped, there is no side effect. Read from an unpopulated subregister, if not specified, will return unpredictable data.

Some of ARF registers are read-only. Write to a read-only ARF register is dropped and there is no side effect.

5.3.4.3 Null Register

ARF Register Type Encoding (RegNum[7:4]):	0000b
Number of Registers:	1
Default Value:	N/A
Normal Access:	N/A
Elements:	N/A
Element Size:	N/A
Element Type:	N/A
Access Granularity:	N/A
Write Mask Granularity:	N/A
SecHalf Control:	N/A
Index-ability:	No

The null register is a special encoding for an operand that does not have physical map. It is primarily used in the instruction to indicate the non-existence of an operand. Write to the null register has no side effect. Read from the null register returns undefined result.

The null register can be used in the place when a source operand is absent. For example, for a single source operand instruction such as MOV, NOT, the second source operand <src1> must be a null register.

When the null register is used as the destination operand of an instruction, it indicates the computed results are not stored in any physical registers. However, implied writes to the accumulator register, if applicable, may still occur for the instruction. When the conditional modifier is present, update to the selected flag register also happens. In this case, the register region fields of the 'null' operand are valid.

Another example use is to use the null register as the posted destination of a *send* instruction for data write to indicate that there is no write completion acknowledgement required. In this case, however, the register region fields are still valid. The null register can also be the first source operand for a send instruction indicating the absent of the implied move. See *send* instruction for details.

5.3.4.4 Address Register

ARF Register Type Encoding (RegNum[7:4]):	0001b
Number of Registers:	1
Default Value:	None
Normal Access:	RW
Elements:	8
Element Size:	16 bits
Element Type:	UW or UD
Access Granularity:	Word
Write Mask Granularity:	Word
SecHalf Control:	N/A
Index-ability:	No

There are eight address subregisters forming an 8-element vector. Each address subregister contains 16 bits. Address subregisters can be used as regular source and destination operands, as the indexing addresses for register-indirect-addressed access of GRF registers, and also as the source of the message descriptor for the *send* instruction.

Table 5-6. Register and Subregister Numbers for Address Register

RegNum[3:0]	SubRegNum[4:0]
0000 = ***a0*** All other encodings are reserved.	When register a0 or subregisters in a0 is used as the address register for register-indirect addressing, the address subregisters must be accessed as unsigned word integers. Therefore, the subregister number field must also be word-aligned. 00000 = **a0.0:uw** 00010 = **a0.1:uw** 00100 = **a0.2:uw** 00110 = **a0.3:uw** 01000 = **a0.4:uw** 01010 = **a0.5:uw** 01100 = **a0.6:uw** 01110 = **a0.7:uw** All other encodings are reserved. However, when register a0 or subregisters in a0 is an explicit source and/or destination register, other data types are allowed as long as the register region falls in the 128-bit range.

Table 5-7. Address Register Fields

Dword	Bits	Subfield Description
3	31:16	**Address subregister a0.7:uw.** This field, with only the lower 12 bits populated, can be used as an unsigned integer for register-indirect register addressing. Format: U12
	15:0	**Address subregister a0.6:uw.** This field, with only the lower 12 bits populated, can be used as an unsigned integer for register-indirect register addressing. Format: U12
2	31:16	**Address subregister a0.5:uw.** This field, with only the lower 12 bits populated, can be used as an unsigned integer for register-indirect register addressing. Format: U12
	15:0	**Address subregister a0.4:uw.** This field, with only the lower 12 bits populated, can be used as an unsigned integer for register-indirect register addressing. Format: U12
1	31:16	**Address subregister a0.3:uw.** This field, with only the lower 12 bits populated, can be used as an unsigned integer for register-indirect register addressing. Format: U12
	15:0	**Address subregister a0.2:uw.** This field, with only the lower 12 bits populated, can be used as an unsigned integer for register-indirect register addressing. Format: U12
0	31:16	**Address subregister a0.1:uw.** This field can be used for register-indirect register addressing or serve as message descriptor for *send* instruction. When used for register-indirect register addressing, it is a 12-bit unsigned integer. For *send* instruction, it provides the higher 16 bits of <desc>. Format: U12 or U16.
	15:0	**Address subregister a0.0:uw.** This field can be used for register-indirect register addressing or serve as message descriptor for *send* instruction. When used for register-indirect register addressing, it is a 12-bit unsigned integer. For *send* instruction, it provides the lower 16 bits of <desc>. Format: U12 or U16.

When used as a source or destination operand, the address subregisters can be accessed individually or as a group. In the following example, the first instruction moves all 8 address subregisters to the first half of GRF register r1, the second instruction replicates a0.4:uw as an unsigned word to the second half of r1, the third instruction moves the first 4 words in r1 into the first 4 address subregisters, and the fourth instruction replicates r1.4 as a unsigned word to the last 4 address subregisters.

```
mov (8) r1.0<1>:uw a0.0<8;8,1>:uw      // r1.n = a0.n for n = 0 to 7 in words
mov (8) r1.8<1>:uw a0.4<0;1,0>:uw      // r1.m = a0.4 for m = 8 to 15 in words
mov (4) a0.0<1>:uw r1.0<4;4,1>:uw      // a0.n = r1.n for n = 0 to 3 in words
mov (4) a0.4<1>:uw r1.4<0;1,0>:uw      // a0.m = r1.4 for m = 4 to 7 in words
```

When used as the register-indirect addressing for GRF registers, the address subregisters can be accessed also individually or in group. When accessed in group, the address subregisters must be group-aligned. For example, when two address subregisters are used for register indirect addressing, they must

be aligned to even address subregisters. In the following example, the first instruction is legal. However, the second one is not. As ExecSize = 8 and the width of <src0> is 4, two address subregisters will be used as row indices, each pointing to 4 data elements spaced by HorzStride = 1 dword. For the first instruction, the two address subregisters are a0.2:uw and a0.3:uw. The two align to a dword group in the address register. However, the two address subregisters for the second instruction are a0.3:uw and a0.4:uw. They are not dword aligned in the address register and therefore violate the above mentioned alignment rule.

mov (8) r1.0<1>:d r[a0.2]<4,1>:d // a0.2 and a0.3 is used for src1

*mov (8) r1.0<1>:d r[a0.**3**]<4,1>:d // **ILLEGAL** use of register indirect*

Implementation restriction: *ISA supports per channel indexing for a source operand. As there are only 8 sub-fields in the address register (to save hardware cost), the execution size of an instruction using per-channel indexing is limited to 8. Software may reload the address register and use compression control **SecHalf** to complete a 16-channel computation.*

Implementation restriction: *When used as the source operand <desc> for the send instruction, only the first dword subregister of a0 register is allowed (i.e. a0.0:ud, which can be viewed as the combination of a0.0:uw and a0.1:uw). In addition, it must be of UD type and in the following form <desc> = a0.0<0;1,0>:ud.*

Implementation restriction: *Elements a0.0 and a0.1 have 16 bits each, but the rest of elements (a0.2:uw through a0.7:uw) only have 12 bits populated each. 12-bit precision supports full indirect-addressing capability for the largest GRF register range. Software must observe the asymmetry of the implementation. When a0.0:uw and a0.1:uw are the source or destination, full 16-bit precision is preserved. However, when a0.2:uw to a0.7:uw are the destination, the higher 4 bits for each element will be dropped; when they are the source, hardware inserts zero to the higher 4 bits for each element.*

Performance Note: *There is only one scoreboard for the whole address register. When a write to some subregisters is in flight, hardware will stall any instruction writing to other subregisters. Software may use the destination dependency control {NoDDChk, NoDDClr} to improve performance in this case. Similarly, when a write to some subregisters is in flight, hardware will stall any instruction sourcing other subregisters until the write retires.*

5.3.4.5 Accumulator Registers

ARF Register Type Encoding (RegNum[7:4]):	0010b
Number of Registers:	2
Default Value:	None
Normal Access:	RW
Elements:	8 or 16
Element Size:	Various
Element Type:	Various
Access Granularity:	Word
Write Mask Granularity:	N/A

SecHalf Control: Yes

Index-ability: No

There are two accumulator registers, *acc0* and *acc1*. They can be accessed either as explicit source and/or destination registers or as implied source and/or destination registers. To a programmer, each accumulator register may contain either 8 doublewords or 16 words of data elements. However, as shown in

Table 5-9, each data element may have higher precision with additional guard bits than that indicated by the numerical data type.

Table 5-8. Register and Subregister Numbers for Accumulate Register

RegNum[3:0]	SubRegNum[4:0]
0000 = ***acc0*** 0001 = ***acc1*** All other encodings are reserved.	Reserved: MBZ The accumulator subfields are individually addressable at word granularity. When an accumulator register is an explicit destination, it follows the rules for a destination register. If an accumulator is an explicit source operand, its register region must match with that of the destination register. The precision for Accumulator for floating point is the exact same as a regular GRF register.

The accumulators will be updated implicitly only if the ***AccWrCtrl*** is on for the current instruction. The accumulator Disable in control register cr0.0 allows software to over-write ***AccWrCtrl*** control for implicit accumulator update. The write enable in word granularity for the instruction will be used to update the Accumulator, the data in the disabled channels will not be updated.

When an accumulator register is used as an implicit source or destination operand, it is acc0 by default. For a SIMD16 DW/float instruction, both acc0 and acc1 are used. If ***SecHalf*** is set, the implicit accumulator is then acc1.

It is illegal to specify different accumulator registers for source and destination operands in an instruction (e.g. "*add (8) acc1:f acc0:f*"). Result of such instruction is unpredictable.

For a SIMD16 DW/float instruction, if an accumulator register is used as an explicit source or destination operand, it must be acc0.

When an accumulator register is used as an explicit source operand, it must be the first source operand <src0>.

Whether the accumulator register is updated for a given instruction depends on several conditions: it can be an explicit destination operand, it can be an implicit destination if the ***AccWrCtrl*** is set in the instruction.

Swizzling is not allowed when accumulator is used as implicit source or explicit source of an instruction.

Implementation Precision Restriction: *As there are only 64 bits per channel in dword mode (D and UD), it is sufficient to store multiplication result of two dword operands as long as the post source modified sources are still within 32 bits. If any one source is type UD and is negated, the negated result*

becomes 33 bits. The dword multiplication results will be 65 bits, bigger than the storage capacity of accumulators. Consequently, the results are unpredictable.

Implementation Precision Restriction: *As there are only 33 bits per channel in word mode (W and UW), it is sufficient to store multiplication result of two word operands with and without source modifier as the result is up to 33 bits. Integer is stored in accumulator in 2's compliment form with bit 32 as the sign bit. As there is no guard bit left, the accumulator can only be sourced once before running into risk of overflowing. When overflow occurs, only modular addition can generate correct result. But in this case, conditional flags may be incorrect. When saturation is used, the output is unpredictable. This is also true for other operations that may result in more than 33 bits of data. For example, adding UDW (FFFFFFFF) with DW (00000001) results in (1FFFFFFFE). The sign bit is now at bit 34 and is lost when stored in the accumulator. When it is read out later from the accumulator, it becomes a negative number as bit 32 now becomes the sign bit.*

Table 5-9. Accumulator Channel Precision

Data Type	# Channel	Bits / Channel	Description
F	8	32	When accumulator is used for float, it has the exact same precision as any GRF register
D (UD)	8	64	When the internal execution data type is doubleword integer, each accumulator register contains 8 channels of (extended) doubleword integer values. The data are always stored in accumulator in 2's compliment form with 64 bits total regardless of the source data type. This is sufficient to construct the 64-bit D or UD multiplication results using an instruction macro sequence consisting mul, mach and shr (or mov).
W (UW)	16	33	When the internal execution data type is word integer, each accumulator register contains 16 channels of (extended) word integer values. The data are always stored in accumulator in 2's compliment form with 33 bits total. This supports single instruction multiplication of two word source in W and/or UW format.
B (UB)	N/A	N/A	Not supported data type.

5.3.4.6 Flag Register [DevSNB]

ARF Register Type Encoding (RegNum[7:4]): 0011b

Number of Registers: 1

Default Value: None

Normal Access: RW

Elements: 2

Element Size: 16 bits

Element Type: UW

Access Granularity: Word

Write Mask Granularity: Word

SecHalf Control: Yes

Index-ability: No

There is one flag register that consists of two 16-bit subregisters. Each flag subregister can be individually addressed. Each bit of a flag subregister corresponds to a data channel. (See Table 3-10 for details). Furthermore, each 16-bit subregister may be split to half with the ***SecHalf*** control in the instruction.

The two flag subregisters (*f0.0:uw* and *f0.1:uw*) can be used as the destination of the conditional modifier and can also be the source of the predication. As both predication flag source and conditional flag destination share the same instruction field, when both are enabled, they use the same flag subregister.

The values held in the individual bits of a flag subregister are the result of the most recent instruction which performed a condition-code evaluation with that flag register specified as the destination of the evaluation result. For example "*add.nz.f0.0 ...*" updates flag f0.0 with the per-channel results of the not-zero condition. The flag subregister has per-bit write enables. When being updated as the secondary destination associated with conditional modifier, only the bits corresponding to the enabled channels in *EMask* are updated. The other bits in the flag subregister are unchanged.

The flag register is not updated for SEL instruction and conditional branch instructions with embedded compares.

The flag register as a whole or as two subregisters can also be an explicit source and/or destination operand.

Table 5-10. Register and Subregister Numbers for Flag Register

RegNum[3:0]	SubRegNum[4:0]
0000 = ***f0:ud*** All other encodings are reserved.	00000 = **f0.0:uw** 00010 = **f0.1:uw** All other encodings are reserved.

Table 5-11. Flag Register Fields

DWord	Bit	Description
0	31:16	**Flag subregister f0.1:uw** Format: U16 FormatDesc Same as f0.0:uw.
	15:0	**Reserved**

5.3.4.7 State Registers

ARF Register Type Encoding (RegNum[7:4]):	0111b
Number of Registers:	1
Default Value:	Provided by the Dispatcher
Normal Access:	RW
Elements:	4
Element Size:	32 bits
Element Type:	UD
Access Granularity:	Byte
Write Mask Granularity:	N/A
SecHalf Control:	No
Index-ability:	No

Thread state registers are read-only registers. They can be accessed as a whole or individually using region-based addressing with byte granularity.

Table 5-12. Register and Subregister Numbers for State Register

RegNum[3:0]	SubRegNum[4:0]
0000 = ***sr0*** All other encodings are reserved.	Valid encoding range: 00000 – 00111 (in unit of byte) For example, 00000 = **sr0.0:ud.** It contains general thread states such as EUID/TID 00100 = **sr0.1:ud.** It contains the additional ID 01000 = **sr0.2:ud.** It contains the Dispatch Mask 01100 = **sr0.3:ud.** It contains the Vector Mask All other encodings are reserved.

Table 5-13. State Register Fields

DWord	Bits	Subfield Description
0 (sr0.0:ud)	31:28	Reserved: MBZ
	27:24	**FFID** (Fixed Function Identifier). Specifies which fixed function unit generates the current thread. This field is set at thread dispatch and is forwarded on the message bus for all out-going messages from this thread.
	23	**Priority Class.** This field, when set, indicates the thread belongs to the high priority class, which has higher scheduling priority over any thread with this field cleared. The priority field below determines the relative priority within the same priority class. This field is initialized by the thread dispatcher at thread dispatch time and stays unchanged throughout the life span of the thread. This field is forwarded on the message bus to the message bus arbiter for all out-going messages. It serves as a priority hint for the target shared function. See shared function chapters whether and how a shared function uses this priority hint. 0 = Low priority class 1 = High priority class
	22:19	Reserved: MBZ
	18:16	**Priority.** This field is the relative aging priority of the thread. This field indicates the 'age' of this thread relative to other thread within the EU. No two threads within the same EU can have the same priority number (independent of the priority class value). Within the same priority class, an older thread (with a larger priority number) has higher schedule priority over a younger thread. This field is set to zero at a thread's dispatch. During a thread's run time, this field may or may not be incremented when a new thread is dispatched to the same EU. It is only incremented when another thread's priority number is incremented and reaches the same value. For example, if currently there is a thread with priority 0 on an EU, dispatching a new thread to this EU will cause the old thread's priority number being incremented to 1. However, if the active thread (assuming for simplicity there is only one) on an EU has a priority number 1 (or 2 or 3), dispatching a new thread to this EU will not change the old thread's priority number. As threads on an EU may terminate in arbitrary orders, the exact number for a thread depends on the dynamic execution of threads.
	15	Reserved: MBZ
	14	CID – The field specifies the Core Identifier for the the EU.
	13	SID – The field specifies the Slice Identifier for the the EU.
	12	HSID – The field specifies the HalfSlice Identifier for the EU.
	11:8	EUID[3:0]. The field specifies the Execution Unit Identifier. The MSB of this field specifies the RowID.
	7:3	Reserved: MBZ
	2:0	**TID** (The thread identifier). Specifies which thread slot the current thread is assigned to. This field is set at thread dispatch.
1 (sr0.1:ud)	31:24	**FFTID**. This field contains the thread ID assigned in fixed functions. There is no connection between FFTID and the TID assigned in the EUs.
	23:0	Reserved : MBZ
2 (sr0.2:ud)	31:0	**Dispatch Mask** (**DMask**). The 32-bit field specifies which channels are active at Dispatch time. This field is used by hardware to initialize the mask register. Format: U32

DWord	Bits	Subfield Description
3 (sr0.3:ud)	31:0	**Vector Mask** (**VMask**). The 32-bit field contains for each 4-bit group the OR of the corresponding 4-bit group in the dispatch mask. This field is used by hardware to initialize the mask register. Format: U32

5.3.4.8 Control Register

ARF Register Type Encoding (RegNum[7:4]):	1000b
Number of Registers:	1
Default Value:	Provided by the Dispatcher
Normal Access:	RW
Elements:	4
Element Size:	32 bits
Element Type:	UD
Access Granularity:	Dword
Write Mask Granularity:	Dword
SecHalf Control:	No
Index-ability:	No

The Control register is a read-write register. It contains four 32-bit subregisters that can be accessed individually.

Subregister *cr0.0*:*ud* contains normal operation control fields such as the floating point mode and the accumulator disable. It also contains the master exception status/control field that allows software to switch back to the application thread from the system routine.

Subregister *cr0.1*:*ud* contains the mask and status/control fields for all exceptions. The exception fields are arranged in significance-decreasing order from MSB to LSB. This allows the system routine to use *lzd* instruction to find the high priority exceptions and handles them first. As each exception is mapped to a single bit, other exception priority order may be implemented by software. System routine may choose to handle one exception at a time, by handle the exception detected by a *lzd* instruction and return to application thread. Or it may choose to handle all the concurrent exceptions, by looping through the exception fields until all outstanding exceptions are handled before returning back to the application thread.

Exception enable bits (bits 15:0 in *cr0.1*:*ud*) control whether an exception will cause hardware to jump to system routine or not. Exception status and control bits (bits 31:16 in *cr*0.1:*ud*) indicate which exceptions have occurred and are used for system routine to clear the exception. Even if a given exception is disabled, the corresponding exception status and control bit still reflects the status whether an exception event has occurred or not.

cr0.2:*ud* contains the **Application IP (AIP)** indicating the current thread IP when an exception occurs.

cr0.3:*ud* is reserved. Writing to this subregister is dropped; result of reading from this subregister is unpredictable.

Fields in Control registers also refer to a virtual register called **System IP (SIP)**. SIP is the virtual register holding the global System IP, which is the initial instruction pointer for the system routine. There is only one SIP for the whole system. It is virtual only from a thread's point of view, as it is not visible (i.e. not readable and not writeable) to the thread software executed on an EU. It can only be accessed indirectly by the hardware to response to exception events. Upon an exception, hardware performs some book keepings (e.g. saving the current IP into AIP) and then jumps to SIP. Upon finishing exception handling, the system routine may return back to the application by clearing the Master Exception Status and Control field in *cr*0, which will cause the hardware to load AIP to IP register. See STATE_SIP command for how to set SIP.

Table 5-14. Register and Subregister Numbers for Control Register

RegNum[3:0]	SubRegNum[4:0]
0000 = ***cr0*** All other encodings are reserved.	00000 = **cr0.0:ud.** It contains general thread control fields 00100 = **cr0.1:ud.** It contains exception status and control 01000 = **cr0.2:ud.** It contains AIP.10100 **(reserved)** All other encodings are reserved.

Table 5-15. Control Register Fields

DWord	Bits	Subfield Description
0 (cr0.0:ud)	31	**Master Exception State and Control.** This field is the master state and control for all exceptions. Reading a 0 indicates that the thread is in normal operation state and a 1 means the thread is in exception handle state. Upon an exception event, hardware sets this field to 1 and switch to SIP. Writing a 1 to this field has no effect. Writing a 0 to this field also has no effect if the previous value is 0. In both cases, the field keeps the previous value. If the previous value of this field is 1, software writing a 0 causes the thread to return to AIP. This transition is automatic – software does not have to move AIP to IP. The value of this field then stays as 0. This field is initialized to 0. 0 = Indicate that the thread is in normal state 1 = Indicate that the thread is in exception state
	30:16	Reserved: MBZ

DWord	Bits	Subfield Description
	15	**Breakpoint Suppress.** This field specifies whether breakpoint exception is suppressed or not. This field is normally set by software and cleared by hardware. If Master Exception Status and Control field is 1, this field is ignored by hardware. If Master Exception Status and Control field is 0 (i.e. not in system routine) and Breakpoint is enabled: If this field is set, breakpoint is temporally ignored (suppressed); Upon a breakpoint condition, the instruction is executed and this bit is automatically reset by hardware. This field is provided to prevent infinite loop of jumping to the system routine on a breakpoint condition. The system routine must set this bit (and also clear the corresponding status and control field) before returning to the application thread. This field has no effect when Breakpoint Enable bits is cleared. This field is initialized to 0. 0 = Breakpoint exception is not suppressed 1 = Breakpoint exception is suppressed
	14:9	Reserved : MBZ
	8	**Reserved**
	7	Reserved : MBZ
	6	Reserved: MBZ
	5:4	Reserved: MBZ
	3	**Vector Mask Enable (VME).** This field indicates DMask or Vmask should be used by EU for execution. This field is set by the Thread Dispatch.. 0 : Use Dispath Mask (DMASK) 1 : Use Vector Mask (VMASK)
	2	**Single Program Flow (SPF).** Specifies whether the thread has a single program flow (SIMDnxm with m = 1) or multiple program flows (SIMDnxm with m > 1). This field affects the operation of all branch instructions. In Single Program Flow mode, all execution channels branch and/or loop identically. This field is initialized by the Thread Dispatch. 0: Multiple Program Flows 1: Single Program Flow **Programming Restrictions:** • **[DevSNB]** The *fork* instruction MUST be used to toggle SPF in the middle of a program to bring the PcIPs to the ExIP. Program is not allowed to write directly into this bit. • Only H1/Q1/N1 are allowed in SPF mode. • **[Errata DevSNB]** IP modification using non-branch instructions are not allowed in SPF mode. **Power Optimization:** If an entire shader doesn't do SIMD branching, driver can set the SPF to 1 to save power in HW.

DWord	Bits	Subfield Description
	1	**Accumulator Disable.** This field controls the update of the accumulator by the instruction field *AccWrCtrl*. If this field is cleared, the accumulator is updated for all instructions with *AccWrCtrl* enabled. If set, the accumulator is disabled for all update operations, maintaining its value prior to being disabled. Setting this field has no effect if the accumulator is the explicit destination operand for an instruction. This field is initialized to 0. 0 = Enable accumulator update 1 = Disable accumulator update *Usage Notes:* *This control bit is primarily designed for the System Routine. That routine is not expected to use the accumulator, though it may need to use instructions which include implicit update of the accumulator. In order to use those instructions within the System Routine, but still preserving the accumulator contents upon return to the application kernel, the System Routine would either (a) save and restore the accumulator, or (b) prevent the accumulator from being unintentionally modified. This control bit has been added for the latter method.* *Software has the option to limit the setting of this control bit strictly within the System Routine. If, by convention, this bit is clear within application kernels, the System Routine can simply set the bit upon entry and clear it prior to returning control to the application kernel. This usage model would not necessarily require cr0.0 to be saved/restored in the System Routine. However, if by convention application kernels are permitted to set this bit, then the System Routine would be required to preserve the content of this bit.* [DevSNB+] Reserved, MBZ.
	0	**Floating Point Mode (FPMode).** This field specifies whether the current floating point operation mode is in IEEE standard mode or the alternative mode. It is used to control the floating operation of the Execution Unit. It is also forwarded on the message sideband for all out-going messages, for example, to control the floating point mode of the Extended Math unit or the Sampler unit. Software may modify this field to dynamically switch between the two floating point modes. This field is initialized by the Thread Dispatch. 0 = IEEE floating point mode 1 = Alternative floating point mode
1 (cr0.1:ud)	31	**Reserved**
	30	**External Halt Exception Status and Control.** This field indicates the External Halt exception. It is set by EU hardware upon receiving the broadcast External Halt signal. System routine should reset this field before returning to application routine in order to avoid infinite loop. This bit may be set or cleared by software. This field is initialized to 0.
	29	**Software Exception Control.** This is the control field of software exception. Setting this field to 1 in application routine will cause an exception. Clearing this field in application routine has no effect. Upon entering system routine, the hardware maintains this field as one to signify software exception. System routine should reset this field before returning to application routine. This field may be set or cleared by software. This field is initialized to 0.

DWord	Bits	Subfield Description
	28	**Illegal Opcode Exception Status.** This field, when set, indicates illegal opcode exception. The exception handle routine normally does not return back to the application thread upon an illegal opcode exception. Leaving this bit set, has no effect on hardware – if system software adversely returns to application routine leaving this field set, it doesn't cause any exception. This field should not be set by software or left set by system routine to avoid confusion. This field is initialized to 0.
	27	**Stack Overflow Exception Status.** This field, when set, indicates stack overflow exception. The exception handle routine normally does not return back to the application thread upon this exception exception. Leaving this bit set, has no effect on hardware – if system software adversely returns to application routine leaving this field set, it doesn't cause any exception. This field should not be set by software or left set by system routine to avoid confusion. This field is initialized to 0.
	23:16	Reserved: MBZ
	15	**Breakpoint Enable.** Specifies whether breakpoint exception is enabled or not. This field is initialized by the Thread Dispatcher. Format = ENABLED 0 = Disabled 1 = Enabled
	14	**Reserved**
	13	**Software Exception Enable.** This field enables or disables the software exception. Enabling or disabling this field may allow host software to turn on/off certain features (such as profiling) without changing the kernel program. This field is initialized by the Thread Dispatcher. Format = ENABLED
	12	**Illegal Opcode Exception Enable.** This field specifies whether illegal opcode exception is enabled or not. Illegal opcode exception includes illegal opcode and undefined opcode, caused by bad program or run time data corruption. This field is initialized by the Thread Dispatcher. Software should normally set it in the interface descriptor. Even though the mechanism is provided to disable illegal opcode exception, it should be used with extreme caution. Format = ENABLED
	11	**Stack Overflow Exception Enable**. This field specifies whether stack overflow exception is enabled or not. The exception includes an overflow or an underflow in the stack space allocated for the thread. This field is initialized by the Thread Dispatcher. Software should normally set this in the interface descriptor. Format = ENABLED
	10:0	Reserved: MBZ

DWord	Bits	Subfield Description
2 (cr0.2:ud)	31:3	**Application IP (AIP).** This is the register storing the instruction pointer before an exception is handled. Upon an exception, hardware automatically saves the current IP into the AIP register, and then sets the **Master Exception State and Control** field to 1, which forces a switch to the System IP (SIP). AIP register may contain either the pointer to the instruction that causes the exception or the one after (such as mask stack overflow/underflow exceptions). This is shown in the following table, where IP is the instruction which generated the exception. **Exception Type** — **AIP Value** Breakpoint — IP External Halt — n/a (1) Software Exception — IP + 1 Illegal Opcode — IP (1) External Halt exception is asynchronous and not associated with an instruction. When the system routine changes the Master Exception State and Control field from 1 to 0. Hardware restores IP from this register. This field is writable allowing returning IP to be altered after an exception handle.
	2:0	Reserved : MBZ

Implementation Restriction on Register Access: *When the control register is used as an explicit source and/or destination, hardware doesn't ensure execution pipeline coherency. Software must set the thread control field to '**switch'** for an instruction that uses control register as an explicit operand. This is important as the control register is an implicit source for most instructions. For example, fields like FPMode and Accumulator Disable control the arithmetic and/or logic instructions. Therefore, if the instruction updating the control register doesn't set 'switch', subsequent instructions may have indeterministic results.*

5.3.4.9 Notification Registers

ARF Register Type Encoding (RegNum[7:4]):	1001b
Number of Registers:	3
Default Value:	No
Normal Access:	RO
Elements:	3
Element Size:	32 bits
Element Type:	UD
Access Granularity:	Dword
Write Mask Granularity:	Dword
SecHalf Control:	No
Index-ability:	No

There are three notification registers (*n0.0:ud*, *n0.1:ud, and n0.2:ud*) used by the *wait* instruction. These registers are read-only and can be accessed in 32-bit granularity.

It should be noted that in the extreme case, it is possible to have more notifications to a thread than the maximal allowable of concurrent threads in the system. Therefore, the range of the thread-to-thread notification count in n0, is larger than the maximum number of threads computed by EUID * TID.

There is only one bit for the host-to-thread notification count in n1.

When directly accessed, this register is read-only. If the value is none zero, the only way to alter the value is to use the wait instruction to decrement the value until zero is reach. A wait instruction on a zero notification subregister will cause the thread to stall, waiting for a notification signal from outside targeting to the same subregister. See wait instruction for details.

Implementation Restrictions: The notification registers are initialized to 0 after hardware/software reset. However, it is not reset at thread dispatch time.

Table 5-16. Register and Subregister Numbers for Notification Register

RegNum[3:0]	SubRegNum[4:0]
0000 = ***n0*** All other encodings are reserved.	00000 = ***n0.0:ud*** 00100 = ***n0.1:ud*** 01000 = ***n0.2:ud*** All other encodings are reserved.

Table 5-17. Fields of Notification Register n0

DWord	Bits	Subfield Description
0	31:16	Reserved: MBZ
	15:0	**Thread to Thread Notification Count.** This register is used by the WAIT instruction for thread-to-thread synchronization. The value read from this register specifies the outstanding notifications received from other threads. It can be changed indirectly by using the WAIT instruction. See WAIT instruction for details. Format: U16

Table 5-18. Fields of Notification Register n1

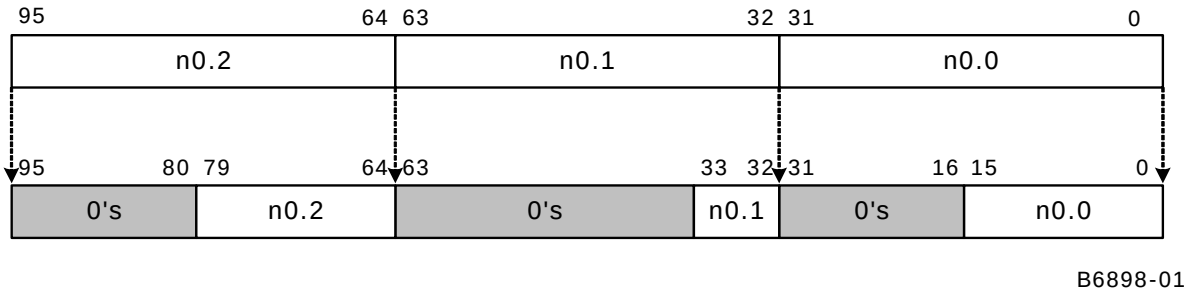
DWord	Bits	Subfield Description
0	31:1	Reserved : MBZ
	0	**Reserved**

Table 5-19. Fields of Notification Register n2

DWord	Bits	Subfield Description
0	31:16	Reserved: MBZ

DWord	Bits	Subfield Description
	15:0	**Thread to Thread Notification Count.** This register is used by the WAIT instruction for thread-to-thread synchronization. The value read from this register specifies the outstanding notifications received from other threads. It can be changed indirectly by using the WAIT instruction. See WAIT instruction for details. Format: U16

Table 5-20. Format of the Notification Register



5.3.4.10 IP Register

ARF Register Type Encoding (RegNum[7:4]):	1010b
Number of Registers:	1
Default Value:	Provided by the Dispatcher
Normal Access:	RW
Elements:	1
Element Size:	32 bits
Element Type:	UD
Access Granularity:	Dword
Write Mask Granularity:	Dword
SecHalf Control:	No
Index-ability:	No

The ip register can be accessed as a 32-bit quantity. It is a read-write register, containing the current instruction pointer, which is relative to the **Generate State Base Address**. Reading this register returns the instruction pointer of the current instruction. The 3 LSBs are always read as zero. Writing this register forces the program flow to jump to the new address. When it is written, the 3 LSBs are dropped by hardware.

[Errata **DevSNB{WA}**] - If IP is updated using a non-flow control instruction by a positive offset, the minimum positive offset must be 2.

[Errata **DevSNB{WA}**] - When SPF is ON, IP may not be updated by non-flow control instructions.

Table 5-21. Register and Subregister Numbers for IP Register

RegNum[3:0]	SubRegNum[4:0]
0000 = ***ip*** All other encodings are reserved.	00000 = **ip:ud** All other encodings are reserved.

Table 5-22. IP Register Fields

DWord	Bits	Subfield Description
0	31:3	**Ip.** Specifies the current instruction pointer. This pointer is relative to the **General State Base Address**.
	2:0	Reserved : MBZ

5.3.4.11 TDR Register

ARF Register Type Encoding (RegNum[7:4]): 1011b

Number of Registers: 8

Default Value: No

Normal Access: RO/CW

Elements: 8

Element Size: 16 bits

Element Type: UW

Access Granularity: Word

Write Mask Granularity: Word

SecHalf Control: No

Index-ability: No

There are 8 thread dependency registers (tdr0.0:uw, tdr0.1:uw, tdr0.2:uw, tdr0.3:uw, tdr0.4:uw, tdr0.5:uw, tdr0.6:uw, and tdr0.7:uw) used by HW for the *sendc* instruction. These registers are read-only and can be accessed in 16-bit granularity.

When accessed explicitly, each thread dependency register has FFTID in the lower 8 bits, bits8 to bits14 are forced to zero by HW. Bit[15] is the valid bit, which indicate whether the current thread has a dependency on the dependency thread stored in this thread dependency register..

The thread dependency registers are read only, the valids can only be set with a thread dispatch, and it will reset by the broadcasting end of thread messages after a thread retired. The FFTID's can only be changed with a therad dispatch. Any write into any of the TDR register will clear the valid bit for the particular TDR if the write enable is true, the FFTID portion is strictly read only.

Table 5-23. Register and Subregister Numbers for TDR Register

RegNum[3:0]	SubRegNum[4:0]
1011 = ***tdr0*** All other encodings are reserved.	00000 = **tdr0.0:uw** 00001 = **tdr0.1:uw** 00010 = **tdr0.2:uw** 00011 = **tdr0.3:uw** 00100 = **tdr0.4:uw** 00101 = **tdr0.5:uw** 00110 = **tdr0.6:uw** 00111 = **tdr0.7:uw** All other encodings are reserved.

5.3.5 Immediate

Two forms of immediate are provided as a source operand: scalar and vector.

For a scalar immediate, it can be of any of the specified numerical data types from a word to a dword. Byte and unsigned byte are not supported as the smallest internal type of the execution pipeline is word. These two numerical types are reserved for future extensions.

The immediate field in an instruction has 32 bits as shown below. For a word or an unsigned word immediate data, software must replicate the same 16-bit immediate value to both the lower word and the high word of the 32-bit immediate field in an instruction.

31	**0**
imm32	

31	**0**
must be the same as [15:0]	**imm16**

The immediate form of vector allows a constant vector to be in-lined in the instruction stream. Both integer and float immediate vectors are supported.

An immediate integer vector is denoted by type v as *imm32:v*, where the 32-bit immediate field is partitioned into 8 4-bit subfields. Each 4-bit subfield contains a signed integer value in 2's compliment form (halfbyte). Therefore each 4-bit subfield has a range of [-8, +7]. This is depicted in the following table.

31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
immV7	immV6	immV5	immV4	immV3	ummV2	immV1	immV0

An immediate float vector is denoted by type **vf** as *imm32:vf*, where the 32-bit immediate field is partitioned into 4 8-bit subfields. Each 8-bit subfield contains a signed float with 3-bit exponent and 4-bit fraction. Each 8-bit subfield provides signed floating point values with restricted range and precision. This is depicted in the following table.

31 24	23 16	15 8	7 0
immVF3	immVF1	immVF1	immVF0

*Restriction: When an immediate vector is used in an instruction, the destination must be 128-bit aligned with destination horizontal stride equivalent to a word for an immediate integer vector (**v**) and equivalent to a dword for an immediate float vector (**vf**).*

5.3.6 Region Parameters

Unlike conventional SIMD architectures where an N-bit wide SIMD instruction can only operate on N-bit aligned SIMD data registers, a region-based register addressing scheme is employed in the architecture. The region-based register addressing capability significantly improves the SIMD computation efficiency by providing per-instruction-based multiple data gathering from register file. This avoids instruction overhead to perform data pack, unpack, and shuffling, which has been observed on other SIMD architectures. One benefit of such capability is allowing various kinds of 3D Graphics API Shader compute models to run efficiently.. Another benefit is allowing high throughput of media applications, which tend to operate on byte or word data elements.

This can be illustrated by the example shown in Figure 5-7 and Figure 5-8. As shown in Figure 5-7, a sequence of SIMD instruction is executed on a conventional load/store based superscalar machine with SIMD instruction extension. The data parallelism can be achieved by first level of loop unrolling. As shown, there is a second level of loop for the task. Before a given SIMD compute instruction, *Process (i)*, can proceed, there might be a load, a data rearrange and a data unpack (and conversion) instruction to load and prepare the input data. After the compute instruction is complete, it might also require pack, re-arrange and store instructions, to format and save the same to memory. At the loop, other scalar computations such as loop count and address generation may be needed. For the same program, when the data can fit in the large GRF register file, the outer loop may be unrolled. Here one or a few block loads (using *send* instruction) may be sufficient to move the working set into GRF. Then the data shuffle can be combined with the processing operation with region-based addressing capability. Per operand float type and mixed data type operation may also allow DevSNB to combine data conditioning operations with computing operations. These techniques in the architecture help to achieve high compute efficiency and throughput for graphics and media applications.

Figure 5-7. Conventional SIMD Instruction Sequence

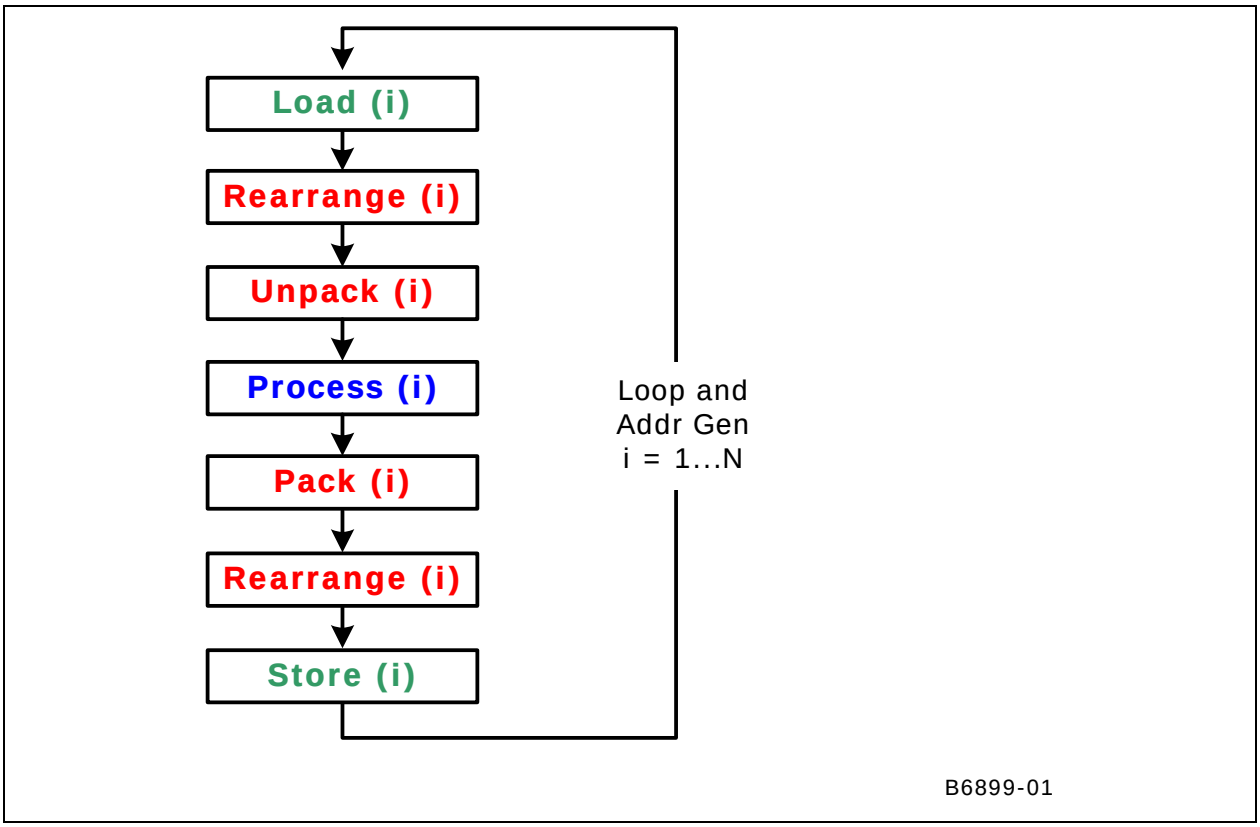
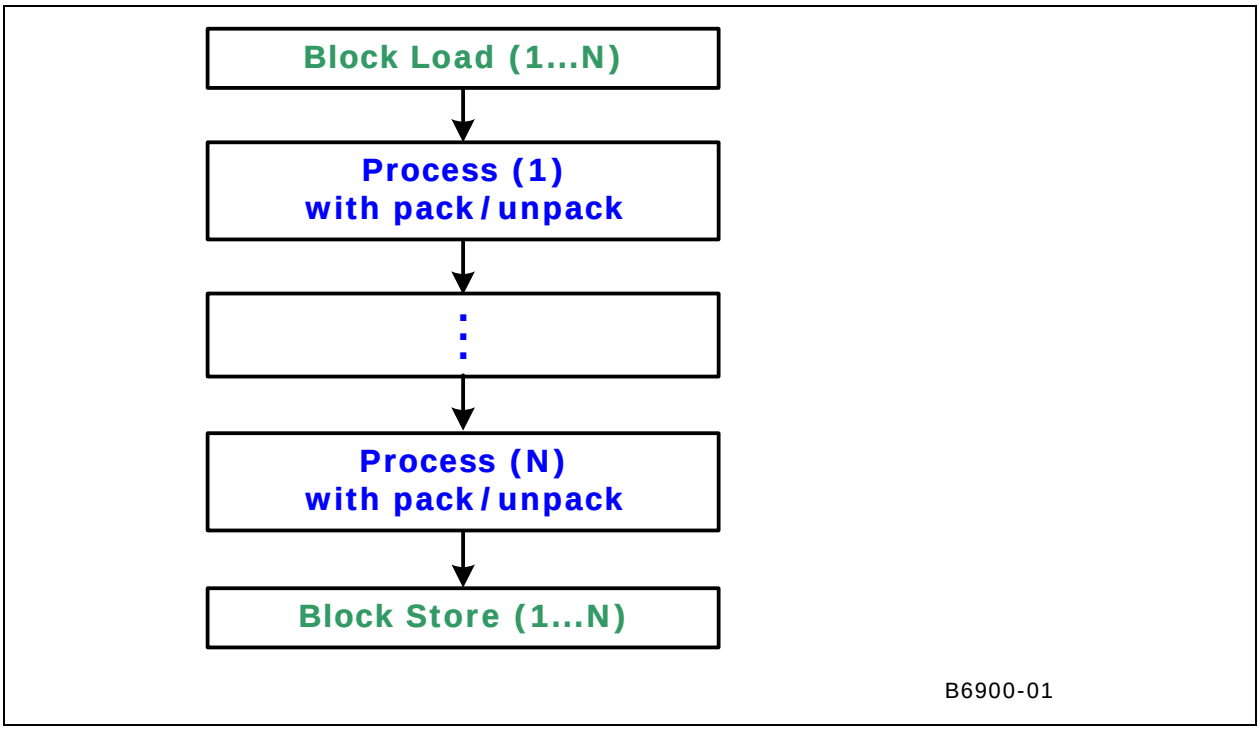


Figure 5-8. DevSNB SIMD Instruction Sequence for the Same Program



In an instruction, each operand defines a region in the register file. A region may contain multiple data elements. Each data element is assigned to an execution channel in the EU. The total number of data elements of a region is called the **size** of the region, or the size of the operand. The number of execution channels is called the **execution size** (***ExecSize***), which is specified in the instruction word. ExecSize determines the size of region for source and destination operands in an instruction.

- For an instruction with two source operands, the sizes of the two source operands must be the same.
- The size of a destination operand generally matches the execution size, therefore equals to the number of source operand(s) in the same instruction.
    - Exception of this rule is present for the integer reduction instructions (such as sad2 and sada2) where the destination area is smaller than the source area.

Regions are **generalized 2-dimensional** (2D) arrays in row-major order. The first dimension is named the **horizontal** dimension (data elements within a row) and the second dimension is termed the **vertical** dimension (data elements in a column). Here, horizontal/vertical and row/column are just symbolic notations. When the GRF or MRF registers are viewed as a row-major 2D array of memory, such a notation normally matches well with the geometric locations of the data elements of an operand. However, as the register region is fully described by the parameters discussed below, the data elements of a register region may not form a regular rectangular shape. For example, Vertical Stride parameter is allowed to be smaller than Horizontal Stride, making the rows of a register region interleave with each other. It should also note that the meanings of horizontal/vertical here is different than that used for the flag control in Section 5.3.4.7.

Specifically, a region-based description of a source operand can take the following format

RegFile RegNum.SubRegNum<VertStride;Width,HorzStride>:type

Parameters are as the follows.

- Register Region Origin (*RegFile*, *RegNum* and *SubRegNum*): This set of parameters, including the register file, *RegFile*, the register number, *RegNum*, and the subregister number, *SubRegNum*, describes the register region origin, which is the location of the first data element of the operand. *RegNum* is in unit of 256-bit and *SubRegNum* is in unit of the data element size.
- Width (*Width*): *Width* specifies the number of data elements along the horizontal dimension, or the number of data elements of a row.
- Horizontal Stride (*HorzStride*): *HorzStride* specifies the step size between two adjacent data elements within a row. It is in unit of data element size, which is determined by the data element *Type*.
- Vertical Stride (*VertStride*): *VertStride* specifies the step size between two adjacent data elements along the vertical dimension (or the step size between two rows). It is again in unit of data element size, which is determined by the data element *Type*.
- Data Element Type (*Type*): *Type* specifies numerical data type (float, word, byte, etc.) of the data elements. All data elements within a region must have the same type.

Both GRF and MRF register files consist of a sequence of 256-bit physical registers. When viewing the register file (GRF for example) as a sequence of 256-bit aligned physical registers, *RegNum* field provides the physical register number, thus for the name. *SubRegNum* provides the sub-field addressing within a physical register. However, when viewing the register file as a byte addressable memory array, (*RegNum* and *SubRegNum*) is just a byte address within the register file with *SubRegNum* providing the lower 5 bits and *RegNum* providing the higher bits.

The execution size is specified for each instruction by the parameter *ExecSize*. The size of the vertical dimension is *ExecSize/Width*, based on the rule that the size of regions must equal to the execution size.

Figure 5-9 depicts the register region description. The example shows a register region of *r4.1<16;8,2>:w*, where the shaded fields denote the data elements in the region and the numbers in these fields are the execution channel assignments. The register region assumes that an *ExecSize* of 16 is set for the instruction. Each data element is a word (as noted by the type field "*:w*"). The origin of the region is at the second word of r4, denoted by *r4.1*. Each row of the region has 8 data elements (words) that are 2 data elements (words) apart. The distance between two rows is 16 words. Note that the region shown is for illustration purpose only. It does not represent a typical usage model nor a performance optimized mode.

Figure 5-9. An example of a register region (*r4.1<16;8,2>:w*) with 16 elements

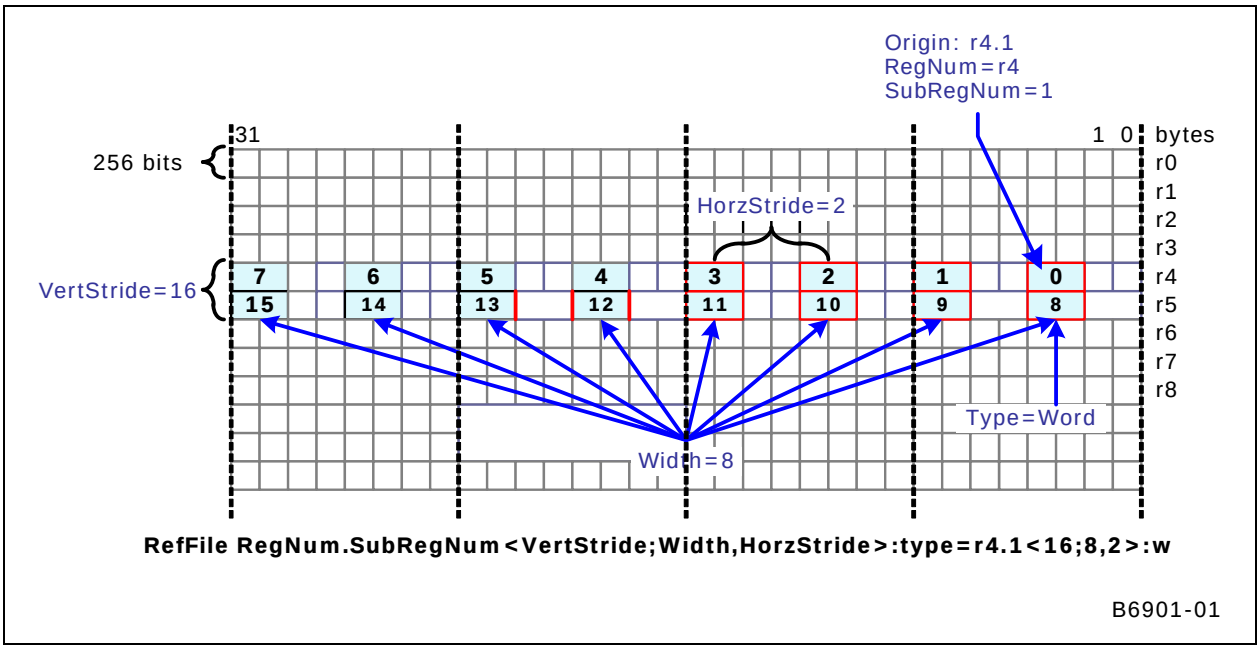
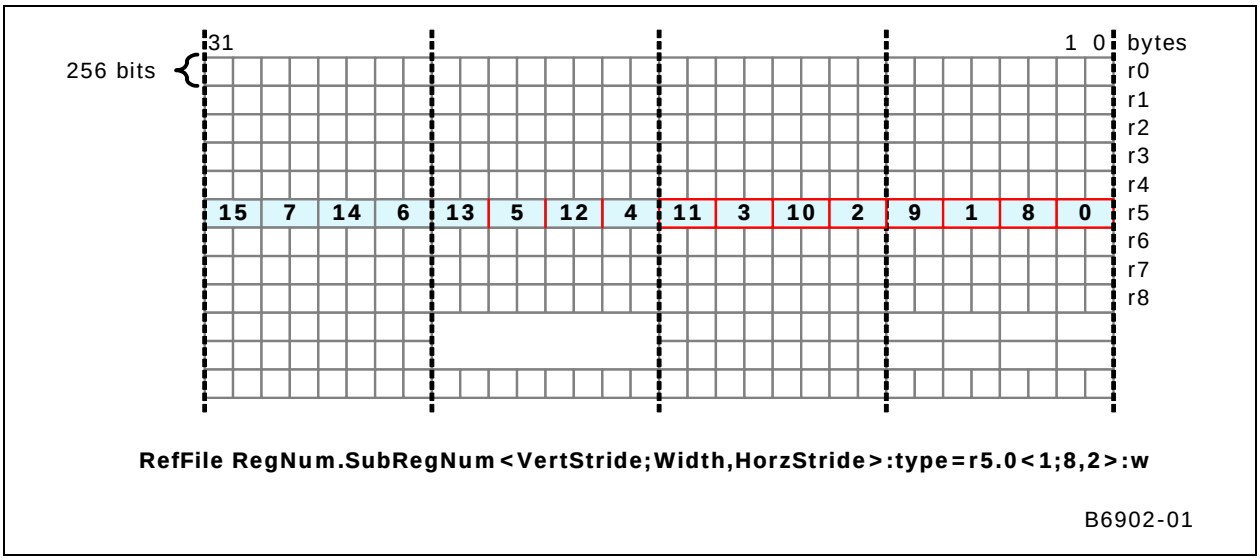


Figure 5-10 shows another example where the rows are interleaved. The region, having word data elements, starts at location r5.0:w. HorzStride, the distance within a row, is 2 words. So the second element (channel number 1) is at location 5.2:w. And there are 8 elements per row. VertStride, the distance between two rows, is only 1 word, which is less than HorzStride. Therefore, the first element of the second row (channel number 8) is at r5.1:w, just next to channel number 0. It is clear from the picture that the two rows are interleaved.

By varying the region parameters, reader may construct other configurations. The next section provides more details on the region-based register addressing. However, there are restrictions imposed by hardware implementation, which can be found in the later sections of this chapter.

Figure 5-10. A 16-element register region with interleaved rows (*r5.0<1;8,2>:w*)



Without considering the source channel swizzle and destination register region description, the above row-major-order region description provides the data assignment to each execution channel. The following pseudo code computes the addresses of data elements assigned to execution channels for a special case when the destination register is aligned to 256-bit register boundary.

```
// Input:          Type: ub | b | uw | w | ud | d | f | v
//                 RegNum: In unit of 256-bit register
//                 SubRegNum: In unit of data element size
//                 ExecSize, Width, VertStride, HorzStride: In unit of data elements
// Output:         Address[0:ExecSize-1] for execution channels

int ElementSize = (Type=="b"||Type=="ub") ? 1 : (Type=="w"|Type=="uw") ? 2 : 4;
int Height = ExecSize / Width;
int Channel = 0;
int RowBase = RegNum<<5 + SubRegNum * ElementSize;
for (int y=0; y<Height; y++) {
        int Offset = RowBase;
        for (int x=0; x<Width; x++) {
                Address [Channel++] = Offset;
```

```
            Offset += HorzStride*ElementSize;
        }
        RowBase += VertStride * ElementSize;
    }
```

As *HorzStride* and *VertStride* are specified independently (note that *VertStride* might be smaller than or equal to *HorzStride*), the region may take various shapes from a replicated scalar, a replicated vector, a vector of replicated scalars, a sliding window, to a non-overlapped 2D array.

A region-based description of a destination operand can take the following simplified format

RegFile RegNum.SubRegNum<HorzStride>:type

The destination operand is only allowed to have a 1 dimensional region. The Register Region Origin and Type are the same as for a source operand. The total number of elements is given by *ExecSize*. However, only *HorzStride* is required to describe the 1D region, not *VertStride* and *Width*.

As a source register region may across multiple physical GRF register, an instruction with such source operands may take more than two execution cycles to gather source data elements for execution. The destination register region is restricted to be within a physical GRF register. In other words, destination scatter writes over multiple physical registers are not supported.

5.3.7 Region Addressing Modes

There are two different register addressing modes: Direct register addressing and register-indirect register addressing. Depending on the register region description, the register-indirect register addressing mode can be further divided into three usages: 1x1 index region where only the origin of register region is provided by the address register, Vx1 index region where the offset of each row of the register region is provided by an address register, VxH index region where the offset of each data element is provided by an address register.

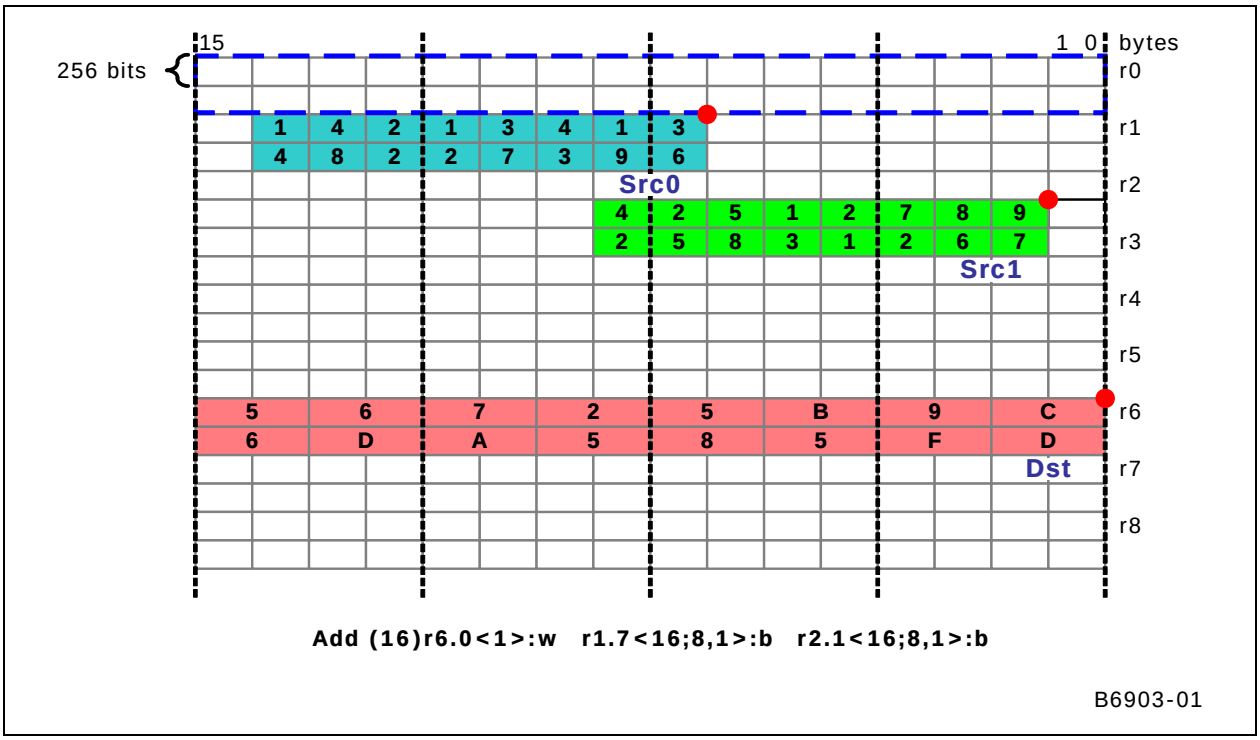
5.3.7.1 Direct Register Addressing

In this mode, all register region parameters are specified for an operand using fields in the instruction word.

Figure 5-11 and Figure 5-12 are two examples of direct register addressing.

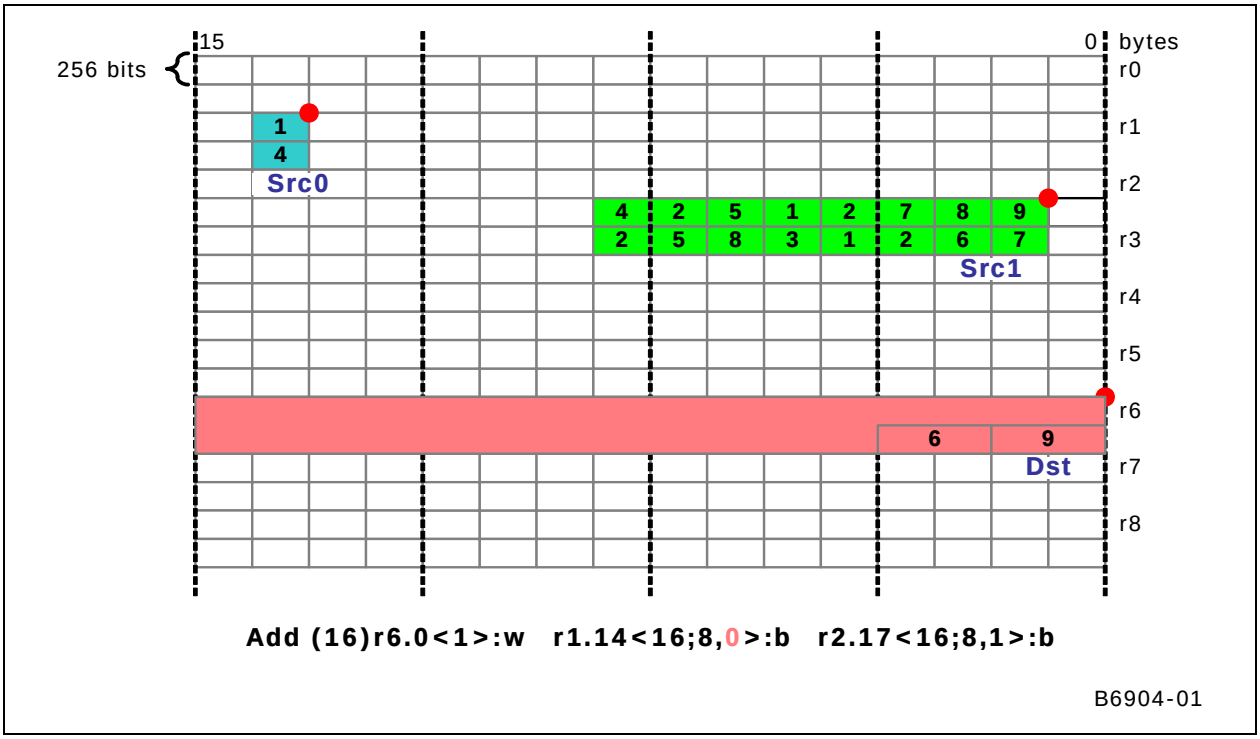
For the example in Figure 5-11, all operands are 2D rectangular regions having the same size of 16 data elements. The two source operands, *Src0* and *Src1*, have 16 bytes. The destination operand, *Dst*, has 16 words. There are 8 elements in a row for *Src0* and *Src1*. The vertical stride of 16 bytes for *Src0* and *Src1* indicates that the first element and the 9'th element are 16 bytes apart in the register file. Note that *Src0* falls into the 256-bit physical GRF register starting at r1.0, but Src1 crosses the 256-bit physical GRF register boundary between r2 and r3. The numbers in the shaded regions are the values of the data elements. Observing the upper right corners of the source/destination regions (first data element), we have C = 3+9.

Figure 5-11. A region description example in direct register addressing



For the example in Figure 5-12, the sizes of areas of *Src0* and *Src1* are the same, but *Src0* contains a vector of replicated scalars. With HorzStride = 0 and Width = 8, the first row of 8 elements in Src0 is a replication of the byte at r1.14. Comparing *ExecSize* of 16 to Width of 8 indicates that there is a second row of 8 elements in *Src0*. With VertStride = 16, the second row in *Src0* is a replication of the byte at r1.20 (20 = 14+16). Effectively, the 16 data elements of *Src0* are {1,1,1,1,1,1,1,1, 4,4,4,4,4,4,4,4}.

Figure 5-12. A region description example in direct register addressing with <src0> as a vector of replicated scalars

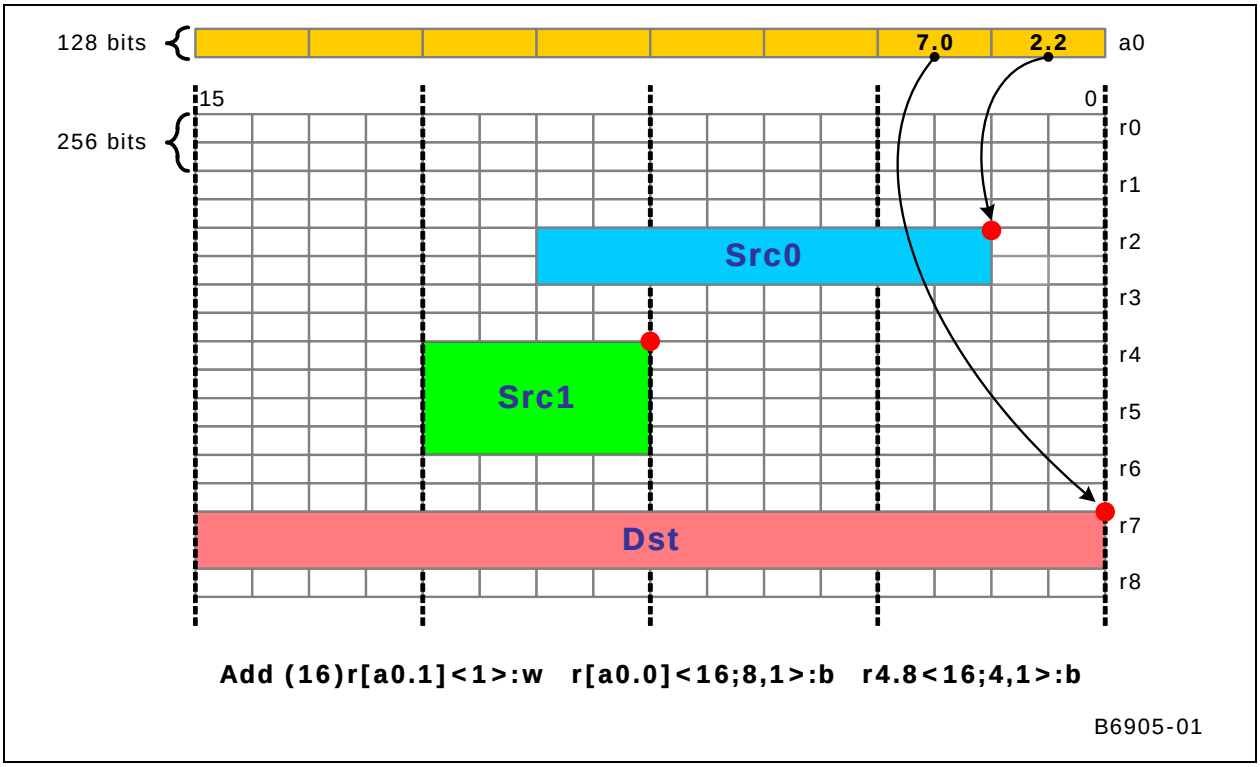


5.3.7.2 Register-indirect Register Addressing with a 1x1 Index Region

In the register-indirect register addressing mode with 1x1 index region, the region origin is provided by the content of the address register, the rest of region parameters are provided by the fields in the instruction word.

Figure 5-13 depicts an example for this addressing mode. For example, the present of full region description <16;8,1> for Src0 indicates that only the origin of the region is provided by the address register a0.0.

Figure 5-13. An example illustrating register-indirect register addressing mode with a 1x1 index region

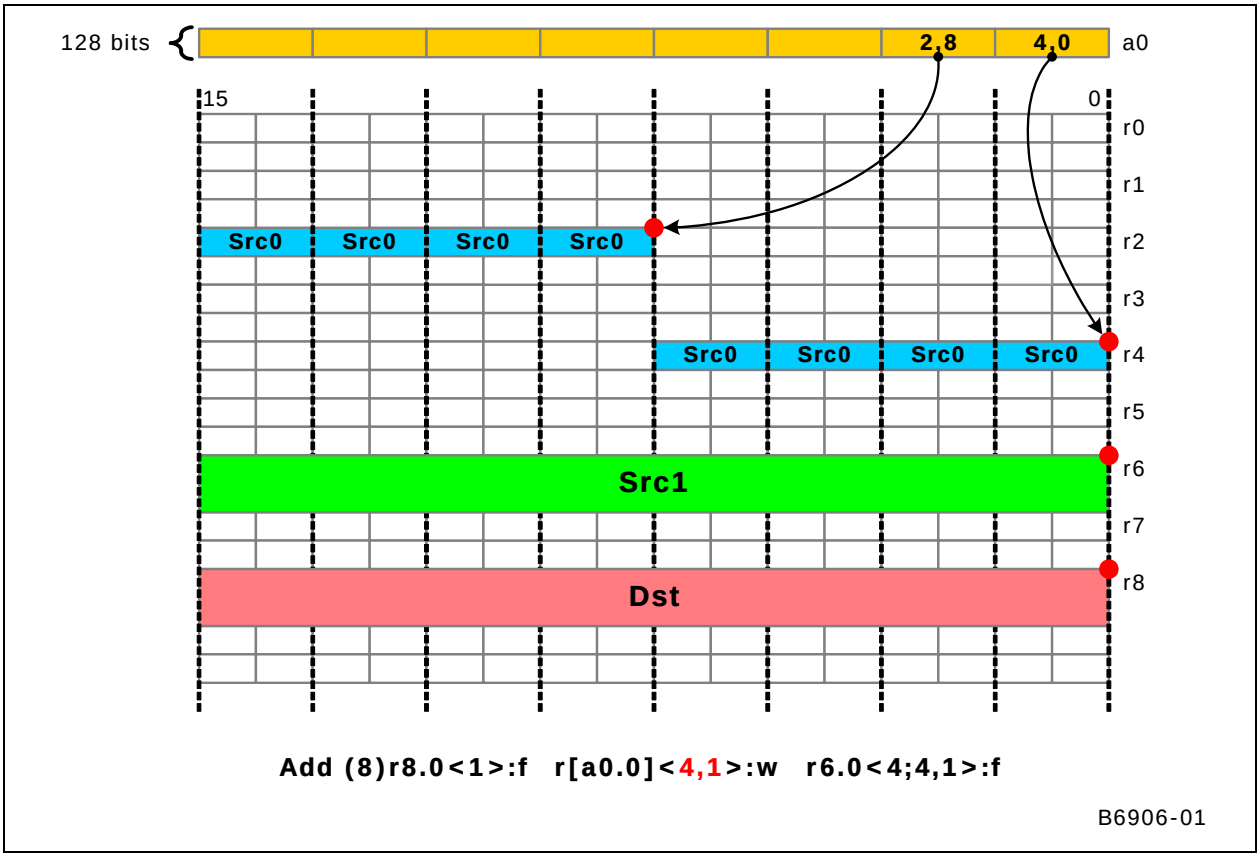


5.3.7.3 Register-indirect Register Addressing with a Vx1 Index Region

In the register-indirect register addressing mode with Vx1 index region, horizontal dimension is described by the fields in the instruction word and the vertical dimension is described by an address register region. Specifically, the origin of each row of the data region is provided by the contents of an address register region. The rows are described by the width and the horizontal stride. The first address register is provided, the following contiguous address registers are for the following rows. The total number of address registers used is inferred by parameters *ExecSize* and *Width.*

An example is provided in Figure 5-14. The assembly syntax notion of a register region without vertical stride, <4,1>, corresponding to the special encoding of vertical stride of 0xF in the instruction word, indicates the VxH or Vx1 mode of indirect register addressing. In this case, the origin for each row of Src0 is provided by the address register. As ExecSize/Width = 2, there are two address registers a0.0 and a0.1, each pointing to a row of 4 data elements.

Figure 5-14. An example illustrating register-indirect-register addressing mode with a Vx1 index region (Src0)

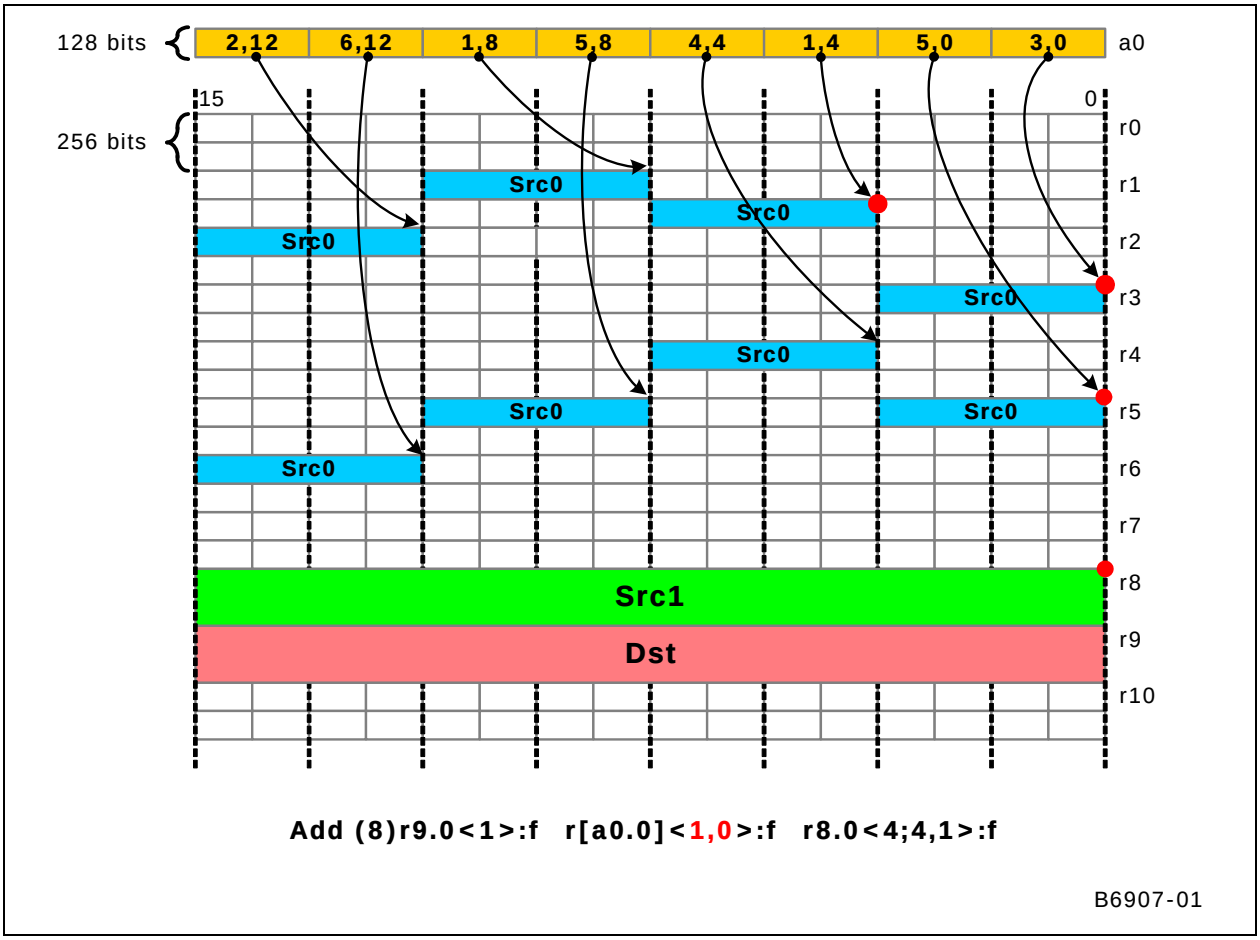


5.3.7.4 Register-indirect Register Addressing with a VxH Index Region

In the register-indirect register addressing mode with VxH index region, the position of each data element is provided by the contexts in an address register region. This mode has the identical syntax as the Vx1 index region mode, and in fact, can be viewed as a special case of the Vx1 mode. When *Width* of the region is 1, the number of address registers used equals *ExecSize*.

An example is provided in Figure 5-15. The absent of vertical stride in the region description <1,0> with width = 1 indicates that the origin for each row of 1 data element of Src0 is provided by the address register. As ExecSize/Width = 8, there are 8 address registers from a0.0 to a0.7, each pointing to a single data elements.

Figure 5-15. An example illustrating register-indirect register addressing mode with a VxH index region (Src0).



5.3.8 Access Modes

There are two basic register access modes controlled by a single bit instruction subfield called Access Mode.

- 16-byte Aligned Access Mode (**align16**): In this mode, the origins of all operands (sources and destination), whether it is by direct addressing or register-indirect addressing, are 16-byte aligned. For example a row in the region description starts at 16-bype aligned and the width the row must be 4 and the 4 data elements within a row must span 16-bytes. In this access mode (and with other restrictions put forward later), full-channel swizzle for both source operands and per-channel mask for destination operand are supported on a 4-component basis. In other words, the control and setting of full source swizzle and destination mask are repeated for every 4 components up to total of *ExecSize* channels.
    - The **align16** access mode can be used for AOS operations. See examples provided in the Primary Usage Model section for SIMD4x2 and SIMD4x1 modes of operation to support 3D API Vertex Shader and Geometric Shader execution.

- 1-byte Aligned Access Mode (**align1**): In this mode, the origins of all operands may be aligned to their data type and could be 1-byte if the operand is of byte type. In this access mode, full region register descriptions are supported, however, source swizzle or destination mask are not supported.
    - The **align1** access mode can be used for SOA operations. See examples provided in the Primary Usage Model section for SIMD8 and SIMD16 modes of operation to support 3D API Pixel Shader. Many media applications also operate well in **align1** access mode.

5.3.9 Execution Data Type

The architecture supports instructions with mixed data types. The internal hardware computation is performed using the execution data type. When an instruction has only one source operand or has two source operands of the same data type, the execution data type is the same as that of the source. When an instruction has two source operands of different types, an execution data type is determined and one of the source operands will be converted to the execution type before the computation is performed. The execution type is independent of the destination data type. When the destination data type is different from the execution data type, a type conversion is performed on the intermediate compute results before the results are written into the destination register. Such a destination type conversion also apply to accumulator registers, implicitly or explicitly.

Determination of the execution data type for two sources of different data types obeys the following rules

- Instuction with mixed float and integer type sources is not allowed.
- Else if any source is a dword, the execution data type is signed dword integer (D)
- Else execution data type is signed word integer (W)

Note that when the execution data type is an integer, it is always a signed integer. This doesn't affect the functional correctness of the instruction as extra precisions are carried within the hardware, including the accumulator. See Instruction Reference Chapter for detailed description for each instruction.

5.3.10 Register Region Restrictions

The following register region rules apply to the implementation. Rules and restrictions for compressed instructions can be found in the Instruction Compression section.

1. *ExecSize* must be equal to or less than the maximum execution size supported for the operand type. As shown in Table 5-24, the maximum execution size is determined by the largest operand type of the sources and destination of the instruction.
2. The mapping of data elements within the region of a source operand is in row-major order and is determined by the region description of the source operand, plus *ExecSize* and destination region description.
3. *ExecSize* must be equal to or greater than *Width*.
4. If *ExecSize* = *Width* and *HorzStride* ≠ 0, *VertStride* must be set to *Width* * *HorzStride*.
5. If *ExecSize* = *Width* but *HorzStride* = 0, there is no restriction on *VertStride*.

6. If *Width* = 1, *HorzStride* must be 0 regardless of the values of *ExecSize* and *VertStride*.

7. If *ExecSize* = *Width* = 1, both *VertStride* and *HorzStride* must be set to zero.

8. If *VertStride* = *HorzStride* = 0, *Width* must be 1 regardless of the value of *ExecSize*.

9. Destination region cannot cross the 256-bit register boundary.

    9.1. Exception to this rule is for a SIMD16 DW/Float instruction where the destination region covers exactly **two adjacent** 256-bit physical registers.

10. Destination region alignment rule.

    10.1. With the exception on 'raw move' described in rule #10.3 and the exception on byte destination in rule #10.5, all destination data elements must be aligned to the size for the execution data type of the instruction. For example, if one of the source operands is in dword mode (a float, a signed or unsigned dword integer), the execution data type will be either float or signed dword integer. Therefore, the destination data elements must be dword aligned. This rule has the following two implications:

        10.1.1. The destination sub-register must be aligned to the size of the execution data type.

        10.1.2. If *ExecSize* is greater than 1, *dst.HorzStride**sizeof*(dst.Type)* must be equal to or greater than the size of the execution data type.

    10.2. If *ExecSize* is 1, *dst.HorzStride* must not be 0. Note that this is relaxed from rule 10.1.2. Also note that this rule for destination horizontal stride is different from that for source as stated in rule #7.

    10.3. When destination type is byte (UB or B), only a 'raw move' using ***mov*** instruction supports packed byte destination register region: *dst.HorzStride* = 1 and *dst.type* = (UB or B). This packed byte destination region is not allowed for any other instructions, including a 'raw move' using ***sel*** instruction. This is because ***sel*** instruction is based on word or dword wide execution channels.

    10.4. When an instruction has a source region that spans two physical registers and destination register contained in one register, one of the followings must be true:

        10.4.1. Destination region is entirely contained in the lower oword of a physical register,

        10.4.2. Destination region is entirely contained in the upper oword of a physical register, or

        10.4.3. Destination elements are evenly split between the two owords of a physical register.

    10.5. When an instruction has a source region that spans two physical registers, the destination spans two physical registers, and the destionation elements are evenly split between the two physical registers.Then each destination register must be entirely derived from one source register.

    10.6. Relaxed alignment rule for byte destination. When destination type is byte (UB or B), destination data elements can be either aligned to the lowest byte or the second lowest byte of the execution channel. For example, if one of the source operands is in word mode (a signed or unsigned word integer), the execution data type will be signed word integer. In this case, the destination data bytes can be either all in the even byte locations or all in the odd byte locations. This rule has the following two implications:

10.6.1. The destination sub-register must be either aligned to the size of the execution data type or one byte higher off the execution data type.

10.6.2. If *ExecSize* is greater than 1, *dst.HorzStride**sizeof*(dst.Type)* must be equal to or greater than the size of the execution data type. This is the same as that in #10.1.2.

11. In Align1 access mode, a source region must be within **two adjacent** 256-bit physical registers.

12. Rules on register-indirect register access:

12.1. An indexed source1 can only have a 1x1 indexed register region – only single index mode is allowed for a source 1.

12.2. An indexed destination can only have a 1x1 indexed register region – only single index mode is allowed for a destination operand.

12.3. Data elements referenced by a single index within a source region cannot cross 256-bit physical register boundary. This applies to register region with a single index or with multiple indices.

12.3.1. A register region with multiple indices may access multiple physical registers as long as data elements associated with each index follow the above-mentioned rule. For example instruction “mov (16) r0.0:uw r[a0.0]<2,2>:uw” is allowed. This is a source gathering instruction whereas the source operand may potentially tough 8 different physical GRF registers.

Table 5-24. Execution size in device hardware

Device	Native Instructions		
Max Operand Size	DWORD	WORD	BYTE
[DevSNB]	16	32	32

Table 5-25. Indirect source addressing support available in device hardware

Device	Indirect Source 0	Indirect Source 1
[DevSNB]	Yes	Yes

5.3.10.1 Examples

Some examples are provided here to illustrate the cases when the register region restrictions are violated. It is provided as informative material to help understanding these restrictions.

Example 1: The following instructions are illegal as they violate rule #10.1, as the destination is not aligned to the execution data type.

```
mov (1) r0.1<1>:b r2.0:w  // dst.SubReg must be even

mov (2) r0.0<1>:b r2.0:w  // dst.HorzStride must be >= 2
```

```
mov (2) r0.0<2>:b r2.0:d  // dst.HorzStride must be >= 4

mov (2) r0.0<2>:b r2.0:f  // dst.HorzStride must be >= 4

mov (1) r0.2<1>:b r2.0:d  // dst.SubReg must be dword aligned
```

Example 2: This instruction is illegal as it violates rule #10.1.2, as when *ExecSize* = 1, *dst.HorzStride* cannot be zero.

```
mov(1) r0.0<0>:b r0.0:d
```

Example 3: This instruction is illegal as it violates rule **#Error! Reference source not found.**, as the source contains one row of 2 elements that spans physical register r2 and r3.

```
mov (2)  r1.0:d r2.7<2;2;1>:d
```

5.3.10.2 Different Raw Moves

Definition of Raw Move: Raw move is an operation that moves data elements from source to destination without altering the bit fields of the data elements. It must use one of the move instructions such as ***mov, sel, movi***. Arithmetic instruction that results in unaltered bit fields of the data elements are not treated as raw move. A raw move may subject to the execution channel enables by using prediction or being present in multi-channel branch code segment. Type conversion by definition cannot be used in a raw move. Therefore, source and destination operands must be of the identical data type. For example, if both source and destination are float, for an arithmetic instruction, denorm will be flushed to zero. However, for a raw move, denorm will be preserved.

Definition of Byte Raw Move: As the minimal execution channel type is word, when the destination stride is greater than one byte, each data element of the source can be mapped to one execution channel. This is referred to as Byte Raw Move. Byte Raw Move allows the destination to be byte aligned, in other words, allowing the destination to not align to execution channels. Byte Raw move subjects to execution channel enables.

Definition of Packed-Byte Raw Move: As the minimal execution channel type is word, when the destination stride is equal to one byte, two data elements of the source are mapped to one execution channel. This is referred to as Packed-Byte Raw Move. Packed-Byte Raw Move allows the destination to be byte aligned, in other words, allowing the destination to not align to execution channels. However, as the data elements are not mapped to execution channels, undefined results may occur if Packed-Byte Raw Move is mixed with execution channel enables. So for Packed-Byte Raw Move, ***WECtrl*** should be used when there are un-enabled channels within the execution size of the instruction.

5.3.11 Destination Operand Description

5.3.11.1 Destination Region Parameters

Based on the above restrictions, a subset of register region parameters are sufficient to describe the destination operand:

- Destination Register Origin
    - Destination Register Number and Destination Subregister Number for direct register addressing mode
    - A Scalar Destination Register Index for register-indirect-register addressing mode
- Destination Register 'Region' – Note that destination register region does not have full region description parameters
    - Destination Horizontal Stride

5.4 SIMD Execution Control

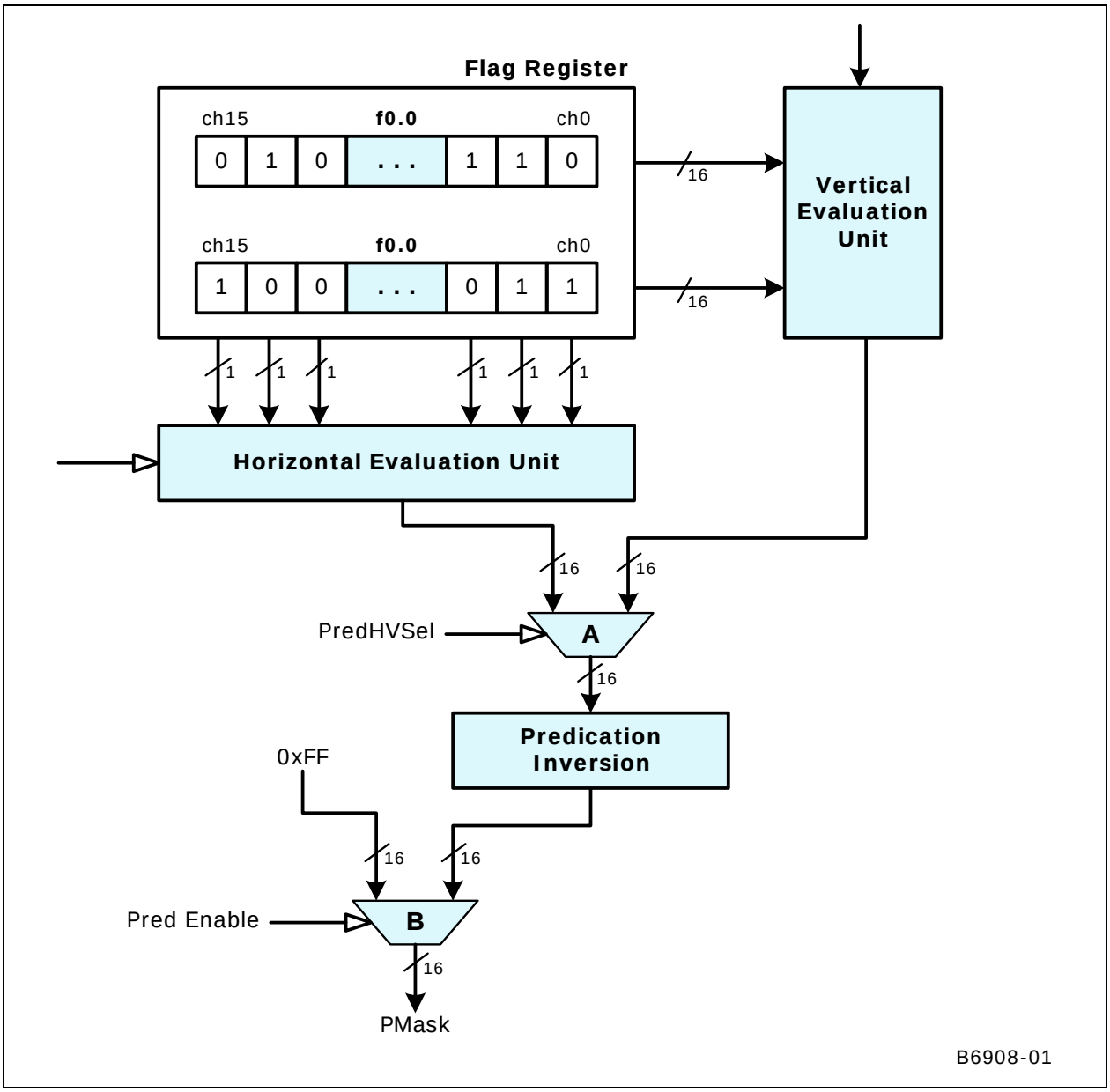
5.4.1 Predication

Predication is the conditional SIMD channel selection for execution on a per instruction basis. It is an efficient way of dynamic SIMD channel enabling without paying branch instruction overhead. When predication is enabled for an instruction, a Predicate Mask (PMask), which contains 16-bit channel enables, is generated internally in EU. Note that PMask is not a software visible register. It is provided here to explain how SIMD execution control works. PMask generation is based on the Predication Control (*PredCtrl*) field, Predication Inversion (*PredInv*) field and the flag source register in the instruction word. See Instruction Summary chapter for definition of these fields.

Figure 5-16 shows the block diagram of the hardware logic to generate PMask. PMask is generated based on combinatory logic operation of the bits in the flag register. Instruction field *PredCtrl* controls the horizontal evaluation unit and vertical evaluation unit. MUX A in the figure selects whether horizontally-evaluated results or vertically-evaluated results are sent to the Predication Invertion unit. The *PredInv* field controls the Prediction Inversion unit. Either one 16-bit flag subregister or the whole flag register may be selected to generate the PMask depending on the predication control modes. MUX B indicates that predication can be enabled and disabled. Predication can be grouped into the following three categories. Predication functionality also depends on the Access Mode of the instruction.

- No predication: Of course, predication can be disabled. This is the most commonly used case.
- Predication with horizontal combination: the predicate mask is generated based on combinatory logic operation of bits within a selected flag subregister.
- Predication with vertical combination: the predicate mask is generated based on combinatory logic operation of bits across flag multiple subregisters.

Figure 5-16. Generation of predication mask



5.4.2 No Predication

When PredCtrl field of a given instruction is set to 0 (“no predication”), it indicates that no predication is applied to this instruction. Effectively, the resulting PMask is all 1’s. This is shown by the 2:1 multiplexer B controlled by the Pred Enable signal in Figure 5-16. Where predication is not enabled for an instruction, multiplex B is selected to output 0xFF to PMask.

5.4.3 Predication with Horizontal Combination

Predication with horizontal combination inputs the 16 bits of a single flag subregister (f0.0:uw or f0.1:uw) and passes them through combinatory logic of the Horizontal Evaluation unit to create PMask.

The simplest combination is 'no combination' – the same 16 bits from selected flag subregister are output to MUX A. In this case, a bit in the selected flag subregister controls the conditional execution of the corresponding execution channel. Let the selected flag subregister be denoted as f0.#, the following pseudo code describes the predicate mask generation for predication with sequential flag channel mapping.

```
If (PredCtrl == "Sequential flag channel mapping") {
        For (ch=0; ch<16; ch++)
                PMask[ch] = (PredInv == TRUE) ? ~f0.#[ch] : f0.#[ch];
}
```

More complex horizontal evaluation is based on channel grouping. A group of adjacent channels (bits from flag subregister) are evaluated together and a single bit is replicated to the group. The size of groups is in power of 2. The supported combination depends on the Access Mode of an instruction.

In **Align16** access mode, horizontal combination is based on 4-channel groups.

- Channel replication: PredCtrl of '.x', '.y', '.z' and '.w' select a single channel from each 4-channel group and replicate it as the output for the group. For example, PredCtrl = '.x' means that channel 0 in each group is replicated.
- OR combination: PredCtrl of '.any4h' means that if **any** of the channel in a group is enabled, outputs for the 4 channels in the group are all enabled.
- AND combination: PredCtrl of '.all4h' means that only when **all** of the channels in a group are enabled, the output for the group is enabled.

These combinations in **Align16** mode can be described by the following pseudo-code.

```
If (Access Mode == Align16) {
        For (ch = 0; ch < 16; ch += 4)
                Switch (PredCtrl) {
                Case '.x':      bTmp = f0.#[ch]; break;
                Case '.y':      bTmp = f0.#[ch+1]; break;
                Case '.z':      bTmp = f0.#[ch+2]; break;
                Case '.w':      bTmp = f0.#[ch+3]; break;
                Case '.any4h':  bTmp = f0.#[ch] | f0.#[ch+1] | f0.#[ch+2] | f0.#[ch+3]; break;
                Case '.all4h':  bTmp = f0.#[ch] & f0.#[ch+1] & f0.#[ch+2] & f0.#[ch+3]; break;
                }
```

```
                bTmp = (PredInv == TRUE) ? ~bTmp : bTmp;
                PMask[ch] = PMask[ch+1] = PMask[ch+2] = PMask[ch+3] = bTmp;
        }
}
```

In **Align1** access mode, horizontal combination is based on AND combination '.any#h' and OR combination '.all#h' on channel groups with various sizes, where # is the number of channels in a group ranging from 2 to 16. This is described by the following pseudo-code.

```
If (Access Mode == Align1) {
        Switch (PredCtrl) {
                Case '.any2h':   groupSize = 2; <op> = '|'; break;
                Case '.all2h':   groupSize = 2; <op> = '&'; break;
                Case '.any4h':   groupSize = 4; <op> = '|'; break;
                Case '.all4h':   groupSize = 4; <op> = '&'; break;
                Case '.any8h':   groupSize = 8; <op> = '|'; break;
                Case '.all8h':   groupSize = 8; <op> = '&'; break;
                Case '.any16h': groupSize = 16; <op> = '|'; break;
                Case '.all16h':  groupSize = 16; <op> = '&'; break;
        }
        For (ch = 0; ch < 16; ch += groupSize) {
                For (inc = 0, bTmp = FALSE; inc < groupSize; inc ++)
                        bTmp = bTmp <op> f0.#[ch+inc];
                For (inc = 0; inc < groupSize; inc ++)
                        PMask[ch+inc] = bTmp;
        }
}
```

5.4.4 Predication with Vertical Combination

Predication with vertical combination uses both flag subregister as inputs. The AND or OR combination is across the subregisters on a channel by channel basis. This is shown by the following pseudo-code.

```
If (Access Mode == Align1) {
        For (ch = 0; ch < 16; ch ++) {
                If (PredCtrl == 'any2v')
                        PMask[ch] = f0.0[ch] | f0.1[ch]
                Else If (PredCtrl == 'any2h')
                        PMask[ch] = f0.0[ch] & f0.1[ch]
        }
}
```

5.5 Instruction Compaction

5.5.1 Motivation and Expected Usage

Instruction Compaction is used to reduce the memory footprint of the shaders. It relys on predefined tables to compact instructions if they has patterns matching the entries inside the compaction table.

Because of the limited size of the compaction table in HW, not all instructions will be compacted by Jitter, HW receives both compacted and noncompacted instructions during execution.

Jitter compacts the instruction using the same compaction table inside HW, Jitter need to calculate any branch/jump offset after instruction compaction is done.

5.5.1.1 Instruction Compaction [DevSNB]

The compaction table is listed below.

Control Bits		
0	00000000000000000	|Align1| ,|(1),|
1	01000000000000000	|Align1| ,|(16),|
2	00110000000000000	|Align1| ,|(8),|
3	00000000100000000	|Align1| |xyzw,|(1),|
4	00010000000000000	|Align1| ,|(2),|
5	00001000100000000	|Align1|-|xyzw,|(1),|
6	00000000100000010	|Align1|NoMask| |xyzw,|(1),|
7	00000000000000010	|Align1|NoMask| ,|(1),|

Control Bits		
8	01000000100000000	|Align1| |xyzw,|(16),|
9	01010000000000000	|Align1| ,|(32),|
10	10110000000000000	|Align1| ,|(8),|sat
11	00100000000000000	|Align1| ,|(4),|
12	11010000000000000	|Align1| ,|(32),|sat
13	11000000000000000	|Align1| ,|(16),|sat
14	01001000100000000	|Align1|-|xyzw,|(16),|
15	01000000000001000	|Align1|NoDDChk| ,|(16),|
16	01000000000000100	|Align1|NoDDClr| ,|(16),|
17	00000000000001000	|Align1|NoDDChk| ,|(1),|
18	00000000000000100	|Align1|NoDDClr| ,|(1),|
19	00111000100000000	|Align1|-|xyzw,|(8),|
20	00001000100000010	|Align1|NoMask|-|xyzw,|(1),|
21	00110000100000000	|Align1| |xyzw,|(8),|
22	00110000000000001	|Align16| ,|(8),|
23	00100000000000001	|Align16| ,|(4),|
24	00110000000000010	|Align1|NoMask| ,|(8),|
25	00110000000000101	|Align16|NoDDClr| ,|(8),|
26	00110000000001001	|Align16|NoDDChk| ,|(8),|
27	00110000000010000	|Align1|SecHalf| ,|(8),|
28	00110000000000011	|Align16|NoMask| ,|(8),|
29	00110000000000100	|Align1|NoDDClr| ,|(8),|
30	00110000100001000	|Align1|NoDDChk| |xyzw,|(8),|
31	00100000000001001	|Align16|NoDDChk| ,|(4),|
Datatypes Bits		
0	001001110000000000	aud|aud|id|<1>|dir
1	001000110000100000	aud|rud|iud|<1>|dir
2	001001110000000001	rud|aud|id|<1>|dir
3	001000000001100000	aud|iud|aud|<1>|dir
4	001010110100101001	ruw|ruw|iuw|<1>|dir
5	001000000110101101	rw|rw|aud|<1>|dir
6	001100011000101100	aw|rub|rub|<1>|dir
7	001011110110101101	rw|rw|iw|<1>|dir

Control Bits		
8	001000000111101100	aw|iw|aud|<1>|dir
9	001000000001100001	rud|iud|aud|<1>|dir
10	001000110010100101	rd|rd|iud|<1>|dir
11	001000000001000001	rud|mud|aud|<1>|dir
12	001000001000110001	rub|rub|aud|<1>|dir
13	001000001000101001	ruw|rub|aud|<1>|dir
14	001000000000100000	aud|rud|aud|<1>|dir
15	001000001000110010	mub|rub|aud|<1>|dir
16	001010010100101001	ruw|ruw|ruw|<1>|dir
17	001011010010100101	rd|rd|rw|<1>|dir
18	001000000110100101	rd|rw|aud|<1>|dir
19	001100011000101001	ruw|rub|rub|<1>|dir
20	001011011000101100	aw|rub|rw|<1>|dir
21	001011010110100101	rd|rw|rw|<1>|dir
22	001011110110100101	rd|rw|iw|<1>|dir
23	001111011110111101	rf|rf|rf|<1>|dir
24	001111011110111100	af|rf|rf|<1>|dir
25	001111011110111101	rf|rf|rf|dir
26	001111011110011101	rf|af|rf|<1>|dir
27	001111011110111110	mf|rf|rf|<1>|dir
28	001000000000100001	rud|rud|aud|<1>|dir
29	001000000000100010	mud|rud|aud|<1>|dir
30	001001111111011101	rf|mf|id|<1>|dir
31	001000001110111110	mf|rf|aud|<1>|dir
Source Operands Bits		
0	000000000000	|dir|<0;1,0>
1	010110001000	|dir|<16;16,1>
2	010001101000	|dir|<8;8,1>
3	001000101000	|dir|<2;2,1>
4	011010010000	|dir|<32;16,2>
5	000100100000	|dir|<1;2,0>
6	010001101100	|indr|<8;8,1>
7	010101110000	|dir|<16;8,2>

Control Bits		
8	011001111000	|dir|<32;8,4>
9	001100101000	|dir|<4;2,1>
10	010110001100	|indr|<16;16,1>
11	001000100000	|dir|<2;2,0>
12	010110001010	-|dir|<16;16,1>
13	000000000010	-|dir|<0;1,0>
14	010101010000	|dir|<16;4,2>
15	010101101000	|dir|<16;8,1>
16	111101001100	|indr|<0;4,1>
17	111100101100	|indr|<0;2,1>
18	011001110000	|dir|<32;8,2>
19	010110001001	(abs)|dir|<16;16,1>
20	010101011000	|dir|<16;4,4>
21	001101001000	|dir|<4;4,1>
22	010000101100	|indr|<8;2,1>
23	010000000000	|dir|<8;1,0>
24	001101110000	|dir|<4;8,2>
25	001100010000	|dir|<4;1,2>
26	001100000000	|dir|<4;1,0>
27	010001101010	-|dir|<8;8,1>
28	001101111000	|dir|<4;8,4>
29	000001110000	|dir|<0;8,2>
30	001100100000	|dir|<4;2,0>
31	001101010000	|dir|<4;4,2>
Subregister Bits		
0	000000000000000	0|0|0|
1	000000000000100	4|0|0|
2	000000110000000	0|12|0|
3	111000000000000	0|0|28|
4	011110000001000	8|0|15|
5	000010000000000	0|0|1|
6	000000000010000	16|0|0|
7	000110000001100	12|0|3|

Control Bits		
8	001000000000000	0|0|4|
9	000001000000000	0|16|0|
10	000001010010100	20|20|0|
11	000000001010110	22|2|0|
12	010000000000000	0|0|8|
13	110000000000000	0|0|24|
14	000100000000000	0|0|2|
15	000000010000000	0|4|0|
16	000000000001000	8|0|0|
17	100000000000000	0|0|16|
18	000001010000000	0|20|0|
19	001010000000000	0|0|5|
20	001100000000000	0|0|6|
21	000000001010100	20|2|0|
22	101101010010100	20|20|22|
23	010100000000000	0|0|10|
24	000000010001111	15|4|0|
25	011000000000000	0|0|12|
26	111110000000000	0|0|31|
27	101000000000000	0|0|20|
28	000000000001111	15|0|0|
29	000100010001111	15|4|2|
30	001000010001111	15|4|4|
31	000110000000000	0|0|3|

5.5.2 Hardware Behavior

Upon receiving an instruction with the bit[29] CompactCtrl bit set, HW uses the 5 indexes inside the compacted instructions to lookup the compaction table, then uses the table output to reconstruct the full size instruction.

If any source of the compacted instruction is immediate, only 13bits of the immediate value is encoded in the compacted instruction, HW sign extends bit[12] all the way for the entire immediate DWord.

Table 5-26. DevSNB Compacted Instruction Format

DW #	Instr Bits Alloc	High Bit	Low Bit	Instr Bits Used	Description	Bits in 128bits Format	Description (Imm. Src0 or Src1)	Bits in 128bits Format (Imm. Src0 or Src1)
	8	63	56	8	*Src1 RegNum*	*[108:101]*	***Imm[23:16] Imm[7:0]***	***[119:112] [103:96]***
	8	55	48	8	*Src0 RegNum*	*[76:69]*	*Src0 RegNum*	*[76:69]*
	8	47	40	8	*Dst RegNum*	*[60:53]*	*Dst RegNum*	*[60:53]*
	5	39	35	5	*Src1Index[4:0]*	*[120:109]*	***Src1Index[4:0]***	***[127:120] [111:104]***
1	3	34	32	3				
	2	31	30	2	*Src0Index[4:0]*	*[88:77]*	*Src0Index[4:0]*	*[88:77]*
	1	29	29	1	*CmptCtrl*	*[29]*	*CmptCtrl*	*[29]*
	1	28	28	1	*FlagSubRegNum*	*[89]*	*FlagSubRegNum*	*[89]*
	4	27	24	4	*CondModifier*	*[27:24]*	*CondModifier*	*[27:24]*
	1	23	23	1	*AccWrCtrl*	*[28]*	*AccWrCtrl*	*[28]*
	5	22	18	5	*SubRegIndex[4:0]*	*[100:96] [68:64] [52:48]*	*SubRegIndex[4:0]*	*[100:96] [68:64] [52:48]*
	5	17	13	5	*DataTypeIndex[4:0]*	*[63:61] [46:32]*	*DataTypeIndex[4:0]*	*[63:61] [46:32]*
	5	12	8	5	*ControlIndex[4:0]*	*[31] [23:8]*	*ControlIndex[4:0]*	*[31] [23:8]*
	1	7	7	1	*DebugCtrl*	*[30]*	*DebugCtrl*	*[30]*
0	7	6	0	7	*Opcode*	*[6:0]*	*Opcode*	*[6:0]*

Definitions of Fields in the Compact Instruction

Bits	Description
63:56	**Bits [108:101] Source1 register number** forms bits [108:101], the source 1 register number field. If immediate source is used, this field forms [103:96] of the 128-bit instruction word.
55:48	**Bits [76:69] Source0 register number** This field, after unpacking, forms bits [76:69], the source 0 register number field, of the 128-bit instruction word.
47:40	**Bits [60:53] Destination register number** This field, after unpacking, forms bits [60:53], the destination register number field, of the 128-bit instruction word.
39:35	**Src1Index** The 5-bit index for source 1. The 12-bit table-look-up result forms bits [120:109], the source 1 register region fields, of the 128-bit instruction word if immediate source is used, this field forms [108:104] of the 128-bit instruction word. Bit[39] is replicated to [127:109] of the 128-bit instruction word.
34:30	**Src0Index** The 5-bit index for source 0. The 12-bit table-look-up result forms bits [88:77], the source 0 register region fields, of the 128-bit instruction word.
29	**CompactCtrl – Compaction Control** This field indicates whether the instruction is in the 64-bit compaction form. When this bit is set (bit 29 of DW0), the instruction length is only 64-bit.. The bit location is fixed in both 128-bit and 64-bit instruction forms. 0 = 128-bit form (normal) 1 = 64-bit compaction form
28	**Bit [89] – FlagSubRegNum** This field, after unpacking, is bit [89] of the 128-bit instruction word.
27:24	**Bits [27:24] – CondModifier** This field, after unpacking, is bits [27:24] of the 128-bit instruction word. The bit location is fixed in both 128-bit and 64-bit instruction forms.

Bits	Description
23	**AccWrCtrl – Implicit Accumulator Write Enable** This field, after unpacking, is bit[28] of the 128-bit instruction word.
22:18	**SubRegIndex** The 5-bit index for sub-register fields. The 15-bit table-look-up result forms bits [100:96], [68,64] and [52,48] of the 128-bit instruction word.
17:13	**DataTypeIndex** The 5-bit index for data type fields. The 18-bit table-look-up result forms bits [63:61] and [46, 32] of the 128-bit instruction word.
12:8	**ControlIndex** The 5-bit index for data type fields. The 17-bit table-look-up result forms bits[31], and [23, 8] of the 128-bit instruction word.
7	**Reserved**
6:0	**Bits [6:0] – Opcode** This field, after unpacking, is bits [6:0] of the 128-bit instruction word. The bit location is fixed in both 128-bit and 64-bit instruction forms.

5.5.3 Rules and Restrictions

In order to reduce the hardware complexity, the following rules and restrictions apply to the compressed instruction:

- Any branch/jump offset need to be based on the physical instruction size, after compaction.
- Any branch/jump instruction with immediate offset larger than 13bits should not be compacted.

5.6 End of Thread

There is no special instruction opcode (such as an END instruction) to cause the thread to terminate execution. Instead, the end of thread is signified by a *send* instruction with the end-of-thread (EOT) sideband bit set. Upon executing a *send* instruction with EOT set, the EU stops on the thread. Upon observing an EOT signal on the output message bus, the Thread Dispatcher makes the thread's resource available. If a thread uses pre-allocated resource managed by a fixed function, such as URB handles and scratch memory, some fixed function protocol also requires the thread to terminate with the message header phase to carry the information in order for the fixed function to release the pre-allocated resource.

EU hardware guarantees that if a terminated thread has in-flight read messages or loads at the time of 'end' that their writebacks will not interfere with either other threads in the system or new threads loaded in the system in the future.

More details can be found in the *send* instruction description in Instruction Reference chapter.

5.7 Creating Conditional Flags

FPU will output 2 sets of conditional signals, 1 set will be generated from before the adder outputs clamping/re-normalizing/format conversion logic, we call this the pre conditional signals. 1 set will be generated from the final results after clamping and re-normalizing/format conversion logic, and we will call this the post conditional signals. The post conditional signals are used for fusing the DX compare instruction. *The flags generated from the post conditional signals should be equivalent to the flags generated by a separate CMP instruction after the current arithmetic instruction.*

The pre conditional signals will be used to generated flags for CMP/CMPN instructions only, this logically does the compare of the 2 input sources. The post conditional signals will be used to generated flags for all the other arithmetic instructions, this logically does the compare of the result with zero.

CMPN with both sources are NaN is a don't care case since this doesn't impact the MIN/MAX operations.

The pre conditional signals include the following:

- **pre_sign** bit: this bit reflects the sign of the computed result directly from the adders, without going through any kind clamping, normalizing, or format conversion logic.
- **pre_zero** bit: this bit reflects whether the value of the adder results are zero, again this should be obtained before any kind clamping, normalizing, or format conversion logic.

The post conditional signals include the following:

- **post_sign** bit: this bit reflects the sign of the final result after all the clamping, normalizing, or format conversion logic.
- **post_zero** bit: this bit reflects whether the value of the adder results are zero after all the clamping, normalizing, or format conversion logic.
- **OF** bit: this bit reflects whether an overflow occured in any of the compution of the current instruction, including clamping, re-normalizing, and format conversion.
- **NC** bit: The NaN computed bit indicates whether the computed result is not a number. It carries valid information for instructions operating on floating point values. For an operation on integer operands, this bit is always set to 0.
- **NS0** bit: The NaN bit indicates whether source 0 of an execution channel is not a number. It carries valid information for instructions operating on floating point values. For an operation on integer operands, this bit is always set to 0.
- **NS1** bit: The NaN bit indicates whether source 1 of an execution channel is not a number. It carries valid information for instructions operating on floating point values. For an operation on integer operands, this bit is always set to 0. For an operation with one source operand, this bit is also set to 0. This bit is only used for the comparison instruction CMPN, which is specifically provided to emulate MIN/MAX operations. For any other instructions, this bit is undefined.

Flag Generation for CMP instructions (The supported Conditional Modifiers are **.e**, **.ne**, **.g**, **.ge**, **.l**, and **.le**.)

Conditional Modifier	Meaning	Resulting Flag Value (for an execution channel)
'**.e**'	Equal-to	**(pre_zero & !(NS0 | NS1)).** This conditional modifier tests whether the 2 sources are equal. If either source is NaN (i.e. NC is true), the flag is force to false.
'**.ne**'	Not-Equal-to	**!(pre_zero & !(NS0 | NS1)).** This conditional modifier test whether the 2 sources are equal. It takes exactly the reverse polarity as modifier '**.e**'.
'**.g**'	Greater-than	**(!pre_sign & !pre_zero & !(NS0 | NS1)).** This conditional modifier tests whether source0 is greater than source1. If either source is a NaN (i.e. NC is true), the flag is forced to false.
'**.ge**'	Greater-than-or-equal-to	**((!pre_sign | pre_zero) & !(NS0 | NS1)).** This conditional modifier tests whether source0 is greater than or equal to source1. If either source is a NaN (i.e. NC is true), the flag is forced to false.
'**.l**'	Less-than	**(pre_sign & !pre_zero & !(NS0 | NS1)).** This conditional modifier tests whether source0 is less than source1. If either source is a NaN (i.e. NC is true), the flag is forced to false.
'**.le**'	Less-than-or-equal-to	**((pre_sign | pre_zero) & !(NS0 | NS1)).** This conditional modifier tests whether source0 is less than or equal to source1. If either source is a NaN (i.e. NC is true), the flag is forced to false.

Flag Generation for CMPN instructions (The supported Conditional Modifiers are **ge**, and **.l**) [DevSNB]

Conditional Modifier	Meaning	Resulting Flag Value (for an execution channel)
'**.ge**'	Greater-than-or-equal-to	**(!pre_sign | pre_zero | (NS1 & (Opcode==CMPN | OPcode==SELwCMod))) & !(NS0 & (Opcode==CMPN)).** This conditional modifier tests whether source0 is greater than or equal to source1. If source-1 is a NaN (i.e. NS is true), the flag is forced to true.
'**.l**'	Less-than	**((pre_sign & !pre_zero) | (NS1 & (Opcode==CMPN | Opcode==SELwCMod))) & !(NS0 & (Opcode==CMPN)).** This conditional modifier tests whether source0 is less than source1. If source-1 is a NaN (i.e. NS is true), the flag is forced to true.

Flag Generation for All Arithmetic Instructions other than CMP/CMPN instructions (The supported Conditional Modifiers are **.e**, **.ne**, **.g**, **.ge**, **.l**, **.le**, **.r**, **.o**, and **.u**.)

Conditional Modifier	Meaning	Resulting Flag Value (for an execution channel)
'**.e**'	Equal-to	**(post_zero & !NC).** This conditional modifier tests whether the 2 sources are equal. If either source is NaN (i.e. NC is true), the flag is force to false.
'**.ne**'	Not-Equal-to	**!(post_zero & !NC).** This conditional modifier test whether the 2 sources are equal. It takes exactly the reverse polarity as modifier '**.e**'.
'**.g**'	Greater-than	**(!post_sign & !post_zero & !NC).** This conditional modifier tests whether source0 is greater than source1. If either source is a NaN (i.e. NC is true), the flag is forced to false.
'**.ge**'	Greater-than-or-equal-to	**((!post_sign | post_zero) & !NC).** This conditional modifier tests whether source0 is greater than or equal to source1. If either source is a NaN (i.e. NC is true), the flag is forced to false.
'**.l**'	Less-than	**(post_sign & !post_zero & !NC).** This conditional modifier tests whether source0 is less than source1. If either source is a NaN (i.e. NC is true), the flag is forced to false.
'**.le**'	Less-than-or-equal-to	**((post_sign | post_zero) & !NC).** This conditional modifier tests whether source0 is less than or equal to source1. If either source is a NaN (i.e. NC is true), the flag is forced to false.
'**.o**'	Overflow	**(OF).** This conditional modifier tests whether the computed result causes overflow – the computed result is outside the range of the destination data type. All other internal conditional signals are ignored.
'**.u**'	Unordered	**(NC).** This conditional modifier tests whether the computed result is a NaN (unordered). All other internal conditional signals are ignored.

5.8 Destination Hazard

The architecture has built-in hardware to avoid destination hazard.

Destination Hazard stands for the risk condition when multiple operations are trying to write to the same destination and the result of the destination may be ambiguous. This may or may not happen on DevSNB for two instructions with the same destination, or with destinations that have overlapped register region, depending on the ordering of the arrival of destination results. Let's consider two instructions in a thread with potential destination hazard. There may be other instruction between them as long as there is no instruction sourcing the same destination. Using register scoreboards, the hardware automatically takes care of the destination hazard by not issuing the second instruction until the destination scoreboard is cleared. However, for certain cases, in fact for most cases, such destination hazard indicated by the register scoreboard is false, causing unnecessary delay of instruction issuing. This may result in lower

performance. The destination dependency control field in the instruction word {*NoDDClr, NoDDhk*} allows software to selectively override such hardware destination dependency mechanism. Such performance optimization hooks must be used with extreme caution. When it is not 100% certainty that it is a false destination hazard, programmer should reply on hardware to result the dependency.

As the destination dependency control field does not apply to *send* instruction, there is only one condition that a programmer may use the {*NoDDClr, NoDDChk*} capability.

- If none of the two instructions is *send*, there CANNOT be any destination hazard. This is because instructions within a thread are dispatched in order (single-issued) and the execution pipeline is in-order and has a fixed latency.

5.9 Non-present Operands

Some instructions do not have two source operands and one destination operand. If an operand is not present for an instruction the operand field in the binary instruction must be filed with null. Otherwise, results are unpredictable.

Specifically, for instructions with a single source, it only uses the first source operand <src0>. In this case, the second source operand <src1> must be set to null and also with the same type as the first source operand <src0>. It is a special case when <src0> is an immediate, as an immediate <src0> uses DW3 of the instruction word, which is normally used by <src1>. In this case, <src1> must be programmed with register file ARF and the same data type as <src0>.

5.10 Instruction Prefetch

Due to prefetch of the instruction stream, the EUs may attempt to access up to 8 instructions (128 bytes) beyond the end of the kernel program – possibly into the next memory page. Although these instructions will not be executed, software must account for the prefetch in order to avoid invalid page access faults. One possible (though inefficient) solution would be to pad the end of all kernel programs with 8 NOOP instructions. A more efficient approach would be to ensure that the page after all kernel programs is at least valid (even if mapped to a dummy page). Note that the **General State Access Upper Bound** field of the STATE_BASE_ADDRESS command can be used to prevent memory accesses past the end of the General State heap (where kernel programs must reside).

6. Exceptions

6.1 Introduction

The architecture defines a basic exception handling mechanism for several exception cases. This mechanism supports both normal operations such as extensions of the mask-stack depth, was well as illegal conditions.

The following exception-types are supported:

Type	Trigger / Source	Sync/Async Recognition
MaskStack Overflow / Underflow	Hardware	Synchronous (w/ special case for 'do'
Software Exception	Thread code	Synchronous
Breakpoint	A bit in the instruction word	Synchronous
Illegal Opcode	Hardware	Synchronous
Halt	MMIO register write	Asynchronous

Threads may choose which exceptions to recognize and which to ignore. This mask information is specified on a per-kernel basis in fixed function state generated by the driver, and delivered to an EU as part of a new-thread dispatch. Upon arrival at the EU, the exception-mask information is used to initialize the exception enable fields of that thread's CR0.1 register, which controls exception recognition. This register is instantiated on a per-thread basis, allowing independent control of exception-type recognition across hardware threads. The exception enables in the CR0.1 register are r/w, and thus can be enabled/disabled via software at anytime during thread execution.

The exception handling mechanism relies on the "system routine", a single subroutine which provides common exception handling for all threads on all EUs in the system. This system routine is defined per-context and is identified via a 32b System-IP (SIP) register in context state. At the time of each context switch, the appropriate SIP for that context is loaded into each EU, allowing each context to have custom implementation of exception handling routines if so desired.

6.2 Exception-Related Architectural Registers

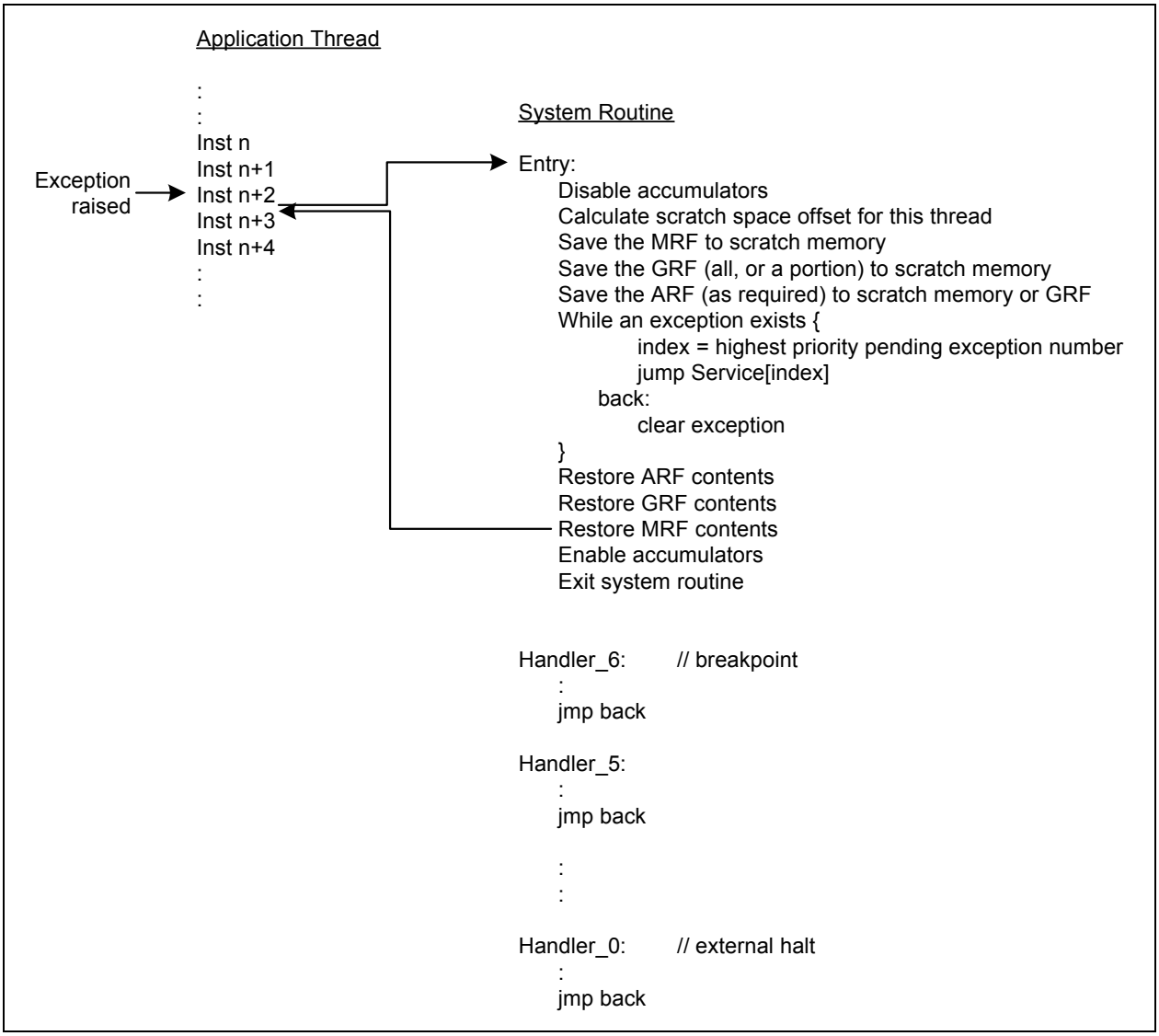
Exception-related registers are defined in architectural register CR0.0 through CR0.2. These registers are instantiated on a per-thread basis providing each hardware thread with unique control over exception recognition and handling. The registers provide the capability to mask exception types, determine the type of raised exception, provide storage the return address, and control exiting from the system routine back to the application thread.

Many of the bits in these registers are manipulated by both hardware and software. In all cases, the read/write operations by hardware and software occur at exclusive times in a thread's lifetime, thus there is no need for an atomic R-M-W operation when accessing these registers.

6.3 System Routine

6.3.1 General Flow of the System Routine

The following diagram illustrates the basic flow of exception handling and structure of the system routine.



6.3.2 Invoking the System Routine

The system routine is invoked in response to a raised exception. Once an exception is raised, no further instructions from the application thread will be issued until the system routine has executed and returned control back to the application thread.

After a exception is recognized by hardware, the EU saves the thread's IP into its AIP register (CR0.2), an then moves the system routine offset, SIP, into the thread's IP register. At this point the next instruction to issue for that thread will be the first instruction of the system routine.

The system routine maintains the same execution priority, GRF and MRF register space, and thread state as that of the application thread from which it was invoked. Due to the assuming the same priority, there may be significant absolute time between exception being raised and the actual invocation of the system routine, as other higher priority threads within the EU continue to execute. From a thread's perspective, once an exception is recognized, the next instruction to issue is from the system routine.

At the time of system routine invocation, there may still be outstanding registers in-flight from the application thread. Depending on the instruction sequence in the system routine, an in-flight register may be referenced by the system routine and cause a register-in-flight dependency. These dependencies are honored by the system routine and may cause the system routine to be suspended until such time that the register retires.

Exception processing is non-nested within an system routine. If a future exception is detected while executing the system routine, the exception is latched into CR0.1, but does not cause a nested re-invocation of the system routine. The exception recognition hardware recognizes only one outstanding exception of each type; i.e. once a specific exception type is detected and latched in CR0.1, and until the exception is cleared, any further exception of that type will be lost.

Accumulators are not natively preserved across the system routine. To make sure the accumulators are in the identical state once control is returned to the application thread, the system routine must either set the Accumulator Disable bit of CR0.0 prior to using any instruction which modifies an accumulator, or manually save/restore the accumulators (to GRF registers or system thread scratch memory) around the system routine. Saving/restoring accumulators, including their extended precision bits, can be accomplished by a short series of mov's and shifts of the accumulator register. Also note the state of the Accumulator Disable bit itself must be preserved unless, by convention, the driver software limits its manipulation to only the system routine.

Further, upon system routine entry, the execution-related masks (Continue, Loop, If, and Active masks, contained in the Mask Register) will remain set as they were in the application thread. Thus only a subset of channels may be active for execution. To enable execution on all channels, the system routine may choose to use the instruction option 'NoMask', or may choose to set the mask registers to the desired value so long as it saves/restores the original masks upon system routine entry/exit.

Similarly there is no hardware mechanism to preserve flags, mask-stacks, or other architectural registers across the system routine. The system routine must ensure that these values are preserved (see Section 6.3.7 for related discussion).

6.3.3 Returning to the Application Thread

Prior to returning control to the application thread, the system routine should clear the proper Exception Status and Control bit in CR1. Failure to do so will force the thread's execution to re-enter the system routine prior to any further instructions being executed from that application thread. (Note that single-stepping functionality is the one exception where the exception's Status and Control bit is not reset prior to exit.)

The system routine may choose to loop under a single invocation of the system routine until all pending exceptions are serviced, or may choose to service exceptions one at a time (a simpler solution, but less efficient).

The system routine is exited, and control returned to the application thread, via a write to the Master Exception State and Control bit of CR0.0. Upon clearing this bit, the value of the AIP architectural register (CR0.2) is restored to the thread's IP register and, with no further exceptions pending, execution resumes that address. The system routine must follow any write to Master Exception State and Control bit with at least one simd-16 'nop' instruction to allow control to transition. Throughout the system routine, the AIP register maintains its value at the time the exception was raised unless directly modified by the system routine. (See the AIP register definition for specifics on the IP value saved to AIP).

6.3.4 System-IP (SIP)

The System IP (SIP) is a 16B-aligned 32b offset of the first instruction of the system routine, relative to the General State Base Address. It is set via the STATE_IP command to the command streamer. The upper 28b of the 32b address is automatically delivered to all EUs.

When the system routine is invoked, the application thread's current IP is first saved into the AIP field of the thread's architectural register CR0.2. The SIP address is then loaded into the thread's IP register and execution continues within the system routine. Thus each invocation of the system routine has a common entry point at the first instruction of the system routine. Upon system routine completion, the value held in AIP is returned to IP and execution continues on the application thread at the place where the exception was recognized.

6.3.5 System Routine Register Space

The system routine uses the same GRF and MRF space at the thread which invoked it. As such all of the calling thread's registers and their contents are visible to the system routine. Further, the system routine must only use r0..r15 of the GRF, as a minimal thread may have requested and been allocated this few. If the system routine requires more registers than this, the driver should establish a higher minimum allocation to all threads. It should also be noted that the system routine may encounter any residual register dependencies of the calling thread until such time that they clear by the return of in-flight writebacks.

Only one 32b GRF location, R0.4, is reserved for system routine usage. This is sufficient to allow the system routine to calculate the appropriate offset of its private scratch memory in the larger system-scratch memory space (as dictated by binding table entry 254). The offset is left as a driver convention, but likely based on a combination of Thread and EU IDs (see example system handler in section 6.3.6). Other than the reserved R0.4 register field, there is no explicit GRF register space dedicated to the system routine, and any GRF needs must be accomplished via: (a) convention between the system routine an application thread, or (b) the system routine temporarily spilling the thread's GRF register contents to scratch memory, and restoration prior to system routine exit.

No persistent storage is natively allocated to the system routine, although a driver implementation may choose to carve out a piece of system scratch memory though it own convention.

Any parameter passing to the system routine (for use by s/w exceptions) is performed via the GRF based on a system-thread/application-thread convention.

6.3.6 System-Scratch Memory Space

There is a single unified system-scratch memory space per context shared by all EUs. It is anticipated that block is further partitioned into a unique scratch sub-space per-thread via convention implemented in the system routine, with a each hardware thread having a uniform block size at a calculated offset from the base address. The block address for a thread is based on an offset derived from the thread's execution unit ID and thread ID made available through the TID and EUID field of architectural register SR0.0.

```
Per_Thread_Block_Size = System_Scratch_Block_Size / (EU_Count * Thread_Per_EU);

Offset = (SR0.0.EID * Threads_Per_EU + SR0.0.TID) * Per_Thread_Block_Size;
```

where in DevSNB...

Threads_Per_EU = 4

EU_Count = 8

System_Scratch_Block_Size is a driver choice

Access to the system-scratch memory is performed through the Data Port via linear single-register or block-based read/write messages. The driver may choose to use any binding table index for system-scratch surface description. As a practical matter, the same index is expected to be used across all binding tables, as the index is typically hard coded in dataport messages used within the system routine coupled with the fact that a single system instance routine is used for all threads. Read/write messages to the Data Port contain the address of the binding table (provided in R0 of all threads) and an offset, from which the Data Port calculates the final target address.

The size of the overall system-scratch memory block is a function of the system routine's feature set, but should be, at a minimum, of size sufficient to cover storing the entire GRF, MRF, flags, and architectural register set for all hardware threads. Additional per-thread memory may also be required for mask-stack spill/fill, system routine constants, and any global persistent storage needs of the system routine.

It is expected that the system-memory block is allocated by the driver at context-create time and remains persistent at a constant memory address throughout the context's lifetime.

6.3.7 Conditional Instructions Within System Routines

It is expected that most, if not all, control flow with in the system routine is scalar in nature. If so, the system routine should set SPF (Single Program Flow, CR0.0) to enable scalar branching. In this mode, conditional/loop instructions do not update the mask-stacks and therefore do not have restrictions on their use nor require the save/restore of hardware mask-stack registers.

If SIMD branching is desired within the system routine, special considerations must be taken. Upon entry to the system routine, the depth of the mask-stacks is unknown at that point, and may be near-full. If so, a subsequent conditional instruction and its associated mask 'push' may cause a stack overflow. This would generate an exception-within-the-system-routine, an unsupported occurrence. To prevent this, if the system routine uses SIMD conditional instructions, it must save the mask-stacks prior to the first SIMD conditional instruction, and restore them after the last SIMD conditional instruction. As a general solution, it may be easiest simply to implement the save/restore as part of the entry/exit code sequence,

using an available GRF register-pair as storage location. Once saved, the stacks should be reset to their empty condition, namely depth = 0 and top-of-stack = 0xFFFFFFFF.

6.3.8 Messages in System Routines

The system routine uses the same MRF space as the thread on whose behalf the system routine was invoked. To allow the thread to resume with the same state as prior to the system routine invocation, the thread's MRF contents must be preserved across a system routine invocation. If the system routine requires MRF space for messages, it must manually save and restore the MRF locations which it uses.

Note that the MRF can only be used as an instruction's destination register, not a source. Therefore there is no option to save the MRF to the GRF. Thus the system routine should save the MRF contents to its dedicated scratch space. By convention it is recommended that MRF register m0 be reserved for system-thread use. This allows the system routine enough space to construct an initial Data Port write message starting at m0 without corrupting any MRF registers, facilitating a complete save/restore of the MRF by the system-thread.

6.3.9 Use of 'NoDDClr'

The instruction word defines an instruction option 'NoDDClr' which overrides the native register dependency clearing mechanism of the typical instruction. When specified, 'NoDDClr' does not clear, at register writeback time, the dependency placed on the destination register of the instruction. Use of this mechanism may provided increased performance when the kernel can guarantee no dependency issues between instructions, but may cause issues with exception handling in some circumstances as discussed here.

Typically 'NoDDClr' is used in an instruction series to enable a sequence of writes to sub-fields of a GRF register without paying a dependency penalty on each instruction. In this case, 'NoDDClr' and 'NoDDChk' are used across an instruction sequence to allow the first instruction to set the destination dependency, interior instructions to write to the GRF register w/o dependency checks, and the last instruction clear the dependency. (This sequence is referred to as a 'NoDDClr' code block going forward). By only allowing the last instruction to clear the dependency, program execution is prevented from going beyond a certain point until all writes of that sequence are known to retire.

The problem arises should an exception be raised within a 'NoDDClr' code block. In this case, there exists the potential for the system routine to hang while attempting to save/restore the code blocks destination register, as the outstanding dependency on that register will not clear until the final instruction of the block is executed – sometime after the system thread returns. Should the system routine attempt to use that register, the system routine will hang waiting on a dependency to clear from an instruction which has not yet been issued.

This is a known condition and will in some cases not allow the full GRF contents to be externally visible in system routine scratch space during a break or halt exception. To minimize the number of cases of such, guidelines are provided below for consideration. (Note that these are general guidelines, some of which can be alleviated through careful coding and register usage conventions and restrictions.)

- 'NoDDClr' code blocks should only be used where absolutely necessary.
- Instructions which may generate exceptions should not be placed within 'NoDDClr' blocks.
- If possible, use 'NoDDClr' on registers high in the thread's register allocation (e.g. r120), thus even if a system routine hang occurs, as much of the GRF is visible as possible. (Note this would also require the system routine to update the progress of the GRF dump, perhaps with each GRF block

written, or to initialize the system routine's scratch space to a known value, to be able to distinguish valid/locations from unwritten locations).

Also a driver implementation may consider a "disable-NoDDclr" option in which jitted code does not use the 'NoDDClr' capability. In this case, there is no change to the code that is jitted other than removal of the 'NoDDClr' instruction option. The code executes as normal, but with a higher number of thread switches in what would have been a NoDDClr code block.

6.4 Exception Descriptions

6.4.1 'Illegal' opcode

The ISA defines a single 'illegal' opcode. The byte value of the 'illegal' opcode is selected to be 0x00 due to it being a likely byte-value encountered by a wayward instruction pointer value. The 'illegal' instruction raises an exception prior to issue and operates as a 'nop' when issued down the execution pipeline. (Specifically, the opcode acts a 'nop', although other non-opcode instruction attributes still apply).

6.4.2 Undefined opcode

All undefined opcodes in the 8b opcode space are detected by hardware. If an undefined opcode is detected, the opcode is overridden by hardware, forcing it to the defined 'illegal' opcode. The offending instruction, should it eventually be issued down the execution unit's pipeline, generates an 'illegal opcode' exception as described in section 6.4.1. Note that the memory location of the offending opcode remains modified and may be queried if desired to determine its original value.

6.4.3 Software Exception

A mechanism is provided to allow an application thread to invoke an exception and is triggered through of the Software Exception Set and Clear bit of CR0.1. Sub-function determination and parameter passing into and out-of the exception handler is left to convention between the system-thread and application-thread. The thread's AIP instruction pointer is incremented prior to system-routine entry, therefore causing execution to resume at the subsequent application-thread instruction when the system routine is exited.

6.4.4 Breakpoint

A single-stepping capability may be implemented by leaving the "Breakpoint Exception Status and Control" set, and clearing the Breakpoint Suppress field prior to system routine exit. This combination causes the instruction associated with the breakpoint to be reissued, this time with the breakpoint suppressed, and then re-entry to the system routine prior to the subsequent instruction due to the lingering breakpoint exception that remained un-cleared.

6.4.5 External Halt

A 'halt' exception may occur via direction manipulation of a MMIO bit by driver software. The halt exception is sent to all EUs simultaneously (although no guarantee is made as to recognition in identical clocks). An EU recognizes this condition internally by generating an External Halt exception. A likely implementation of a handling routine would dump the thread's state to programmer-visible memory (such as the system routine's scratch space) for inspection purposes. Although generally recognized within a

few clocks, there is no specification as to the latency between triggering the Halt condition and it being recognized by an EU.

6.5 Events Which Do Not Generate Exceptions

The following conditions are either not recognized or do not generate an exception.

Illegal Instruction Format

This includes malformed instructions in which the opcode is legal, but the source or destination operands, or instruction attributes are not compliant with the instruction specification. There is no direct hardware support to detect these cases and the outcome of issuing a malformed instruction is undefined. Note that DevSNB does not support self-modifying code, therefore the driver has an opportunity to detect such cases before the thread is placed in service.

Malformed Message

A messages contents, destination registers, lengths, and descriptors are not interpreted in anyway by the execution units. Errors in specifying any of these fields do not raise exceptions in the execution unit but may be detected and reported by the shared functions.

GRF Register Out-of-Bounds

Unique GRF storage is allocated to each thread which, at a minimum, satisfies that the register requirements specified in the thread's declaration. References to GRF register numbers beyond that called for in the thread's declaration do not generate exceptions. Depending on implementation, out-of-bounds register numbers may be remapped to r0..r15, although this functionality should not be relied upon by the thread. The hardware guarantees the isolation of each threads register space, thus there is no possibility of direct register manipulation from an out-of-bounds register access.

MRF Register Out-of-Bounds

A fixed amount of MRF register space is allocated for each thread, namely m0 through m23. References to MRF registers beyond m23 do not generate exceptions. Depending on implementation details, out-of-bounds register numbers may alias to in-bounds register numbers, although this functionality should not be relied upon by the thread.

Hung Thread

There is no hardware mechanism in the execution units to detect a hung thread, and should it occur, the thread remains hung indefinitely. It is the expectation that one or more hung threads will eventually cause the driver to recognize a context timeout and take appropriate recovery action.

Instruction Fetch Out of Bounds

The EUs implement a full 32b instruction address range (with the 4 lsb's don't care), making it possible for a thread to attempt to jump to any 16B aligned offset in the 32b address space. The EU itself does not provide any type of address checking on its instruction request stream sent to the memory/cache hierarchy, although various memory address related error conditions are reported through the Memory Interface Registers (specifically "Page Table Error Register").

FPU Math Errors

The EU's floating point units have defined behavior for traditional floating point errors and do not generate exceptions. Therefore there is no support for signaling FPU math errors as exceptions.

Destination Register Overflow

Depending on source operand contents, destination register size, and operation being performed, overflows may occur in the EU's pipeline. These are not flagged as exceptions and software must explicitly check the overflow bit in the thread's architectural register if overflow is a concern.

6.6 System Handler Example

The following code sequence illustrates some concepts of the system routine. It is intended to be just a shell, without getting into the specifics of each exception handler. The example frees enough MRF and GRF space to get the routine started, then jumps to the handler for the specific exception. Many other implementations are also valid, including single exception servicing (as opposed to looping) per invocation, and saving only the GRF or MRF space required by the exception being serviced.

```
#define ACC_DISABLE_MASK                0xFFFFFFFD
#define MASTER_EXCP_MASK                0x7FFFFFFF
#define SYSROUTINE_SCRATCH_BLKSIZE      16384          //for example

// --- SharedFunc IDs ---
#define DPR                             0x04000000
#define DPW                             0x05000000

// --- message lengths ---
#define ML5                             0x00500000
#define ML9                             0x00900000

// --- response lengths ---
#define RL0                             0x00000000
#define RL4                             0x00040000
#define RL8                             0x00080000

// --- dataport block sizes ---
#define BS1_LOW                         0x0000
#define BS1_HIGH                        0x0100
#define BS2                             0x0200
#define BS4                             0x0300
```

```
// --- Scratch Layout ---
#define SCR_OFFSET_MRF          0
#define SCR_OFFSET_GRF          512             // + 16 reg
#define SCR_OFFSET_ARF          512 + 4096      // + 16 + 128 reg


// --- Write Dataport constants ---
// target=dcache, type= oword_block_wr, binding_tbl_offset=0
#define DPW                     0x000

// --- Read Dataport constants ---
// target=dcache, type= oword_block_rd, binding_tbl_offset=0
#define DPR                     0x000

Sys_Entry:
// --- calc scratch offset for this thread into r0.4 ---
shr   (1) r0.4 sr0.0:uw 6                                   {NoMask}
add   (1) r0.4 r0.4 sr0.0:ub                                {NoMask}
mul   (1) r0.4 r0.4 SYSROUTINE_SCRATCH_BLKSIZE      {NoMask}

// --- setup m0 w/ block offset
mov   (8) m0 r0                                             {NoMask}

// --- save mrf 7...0; (may choose to save the whole mrf)
add   (1) m0.2 r0.4 SCR_OFFSET_MRF                          {NoMask}
send  (8) null m0 null DPW|ML9|RL0                          {NoMask}

// --- save mrf 8...15; (optional; req'ed if sys-routine stays w/in mrf7-0)
mov   (8) m7 r0                                             {NoMask}
add   (1) m7.2 r0.4 (SCR_OFFSET_MRF + 256)      {NoMask}
send  (8) null m7 null DPW|ML9|RL0                          {NoMask}
```

```
// --- save r0..r1 to system scratch ---
// --- (Note: done as a single register to guarantee external
// ---  visibility — see “Use of ‘NoDDClr’” in Excpetions Bspec chapter
mov  (16)  m1 r0                                        {NoMask}
send (8)  m0 null null DPW|ML2|RL0                      {NoMask}

// --- save r2..r3 to free some room
mov  (16) m3 r2                                         {NoMask}
add  (1)  m0.2 r0.4 SCR_OFFSET_GRF + 64                 {NoMask}
send (8)  m0 null null DPW|ML4|RL0                      {NoMask}

// --- save r4..r7 to free some room (optional, depending on needs)
mov  (16) m8 r4                                         {NoMask}
mov  (16) m10 r6                                        {NoMask}
add  (1)  m7.2 r0.4 (SCR_OFFSET_GRF + 128)      {NoMask}
send (8)  m7 null null DPW|ML5|RL0                      {NoMask}

// --- save r8..r11 to free some room (optional, depending on needs)
mov  (16) m1 r8                                         {NoMask}
mov  (16) m3 r10                                        {NoMask}
add  (1)  m0.2 r0.4 (SCR_OFFSET_GRF + 256)      {NoMask}
send (8)  m0 null null DPW|ML5|RL0                      {NoMask}

// --- save r12..r15 to free some room                 (optional, depending on needs)
mov  (16) m8 r12                                        {NoMask}
mov  (16) m10 r14                                       {NoMask}
add  (1)  m7.2 r0.4 (SCR_OFFSET_GRF + 384)      {NoMask}
send (8)  m7 null null DPW|ML5|RL0                      {NoMask}

// --- save ARF registers (optional, depending on use) ---
// flags, maskstacks, others...
```

```
            // --- save f0.0 ---
             mov (1)  r1.0:uw f0.0                                  {NoMask}

Next:       // --- exceptions pending? If not, exit ---
            cmp.e (1) f0.0 cr0.4:uw 0:uw                           {NoMask}
            (f0.0) mov (1) IP EXIT                                 {NoMask}

            // --- find highest priority exception ---
            lzd  (1) r1.1:uw cr0.4:uw                              {NoMask}

            // --- jumptable to service routine ---
            jmpi (1)  r1.1:uw                                      {NoMask}
            mov  (1)  IP CRService_0                               {NoMask}
            mov  (1)  IP CRService_1                               {NoMask}
            mov  (1)  IP CRService_2                               {NoMask}
            // :
            // :
            // :
            mov  (1)  IP CRService_15                              {NoMask}

            mov  (1)  IP Next
Service_0:
            // clear exception from CR0.1
            // perform service routine
            // jump to exit (or if looping on exceptions, go to next loop)

            // :
            // :

Service_15:
            // clear exception from CR0.1
```

```
        // perform service routine
        // jump to exit (or if looping on exceptions, go to next loop)

Exit:
        // --- restore f0.0 ---

        // --- restore ARF registers (as required) ---
        // flags, maskstacks, others...

        // --- restore r12..r15 ---
        // --- restore r8..r11 ---
        // --- restore r4..r7 ---
        // --- restore r0..r3 ---

        // --- restore m8..m15 ---
        // --- restore m0..m7 ---

        // --- clear Master Exception State bit in CR0.0
        and  (1)  cr0.0 cr0.0 MASTER_EXCP_MASK
        nop  (16)
```

Below is a code sequence to programmatically clear the GRF scoreboard in the case of a timeout waiting on a register that may never return:

```
// At this point, all we know is we have a hung thread.  To get around
        // any hung dependency, we can walk the GRF using NoDDChk, using execution mask
        // of f0 = 0 so we don't touch the register contents.

Clear_Dep:
        mov f0  0x00
        (f0) mov r0 0x00 {NoDDChk}
        (f0) mov r1 0x00 {NoDDChk}
        (f0) mov r2 0x00 {NoDDChk}
            ...
            ...
        (f0) mov r127 0x00 {NoDDChk}

        // GRF scoreboard now cleared.
```

7. Instruction Set Summary

7.1 Instruction Set Characteristics

7.1.1 SIMD Instructions and SIMD Width

Instructions are SIMD (single instruction multiple data) instructions. The number of data elements per instruction, or the execution size, depends on the data type. For example, the execution size for instructions operating on 256-bit wide vectors can be up to 8 for 32-bit data types, and be up to 16 for 16-bit data. The maximum execution size for instructions for 8-bit data types is also limited to 16.

An instruction compression mode is supported for 32-bit instructions (including mixed 32-bit and 16-bit data computation). A compressed instruction works on twice as many SIMD data as that for a non-compressed instruction. Non-compressed instructions are also referred to as 'native' instructions. A compressed instruction is converted into two native instructions by the instruction dispatcher in the EU.

Instructions are executed on a narrower SIMD execution pipeline. Therefore, native instructions take multiple execution cycles to complete. See Table 7-1 for parameters for difference device hardware.

Table 7-1. GRF instruction execution parameters in device hardware

Device	Execution Pipeline	Native Instruction	
	Width (bits)	Max Width (bits)	Min Execution Cycles (clocks)
[DevSNB]	**128**	**512**	**4**

7.1.2 Instruction Operands and Register Regions

The majority of instructions may have up to three operands, two sources and one destination. Each operand is able to address a register region. Source operands support negate and absolute modifier and channel swizzle, and the destination operand supports channel mask.

Dual destination instructions are also supported (four-operand instructions in a general sense): One case is for the implied destination – flag register, where the conditional modifiers and the predicate modifiers may apply. Another case is the message header creation (implied move or implied assembling of the header) in the *send* instruction.

Each execution channel contains an accumulator that is wider than the input data to support back-to-back accumulation operations with increased precision. The added precision (see accumulator register description in Execution Environment chapter) determines the maximum number of accumulations before possible overflow. The accumulator can be pre-loaded through the use of ***mov***. It can also be pre-loaded by arithmetic instructions such as ***add***, ***mul***, since the result of these instructions can go to the accumulator. The accumulator registers are per thread and therefore safe for thread switching.

Register access can be direct or register-indirect. Register-indirect register access uses address registers plus an immediate offset term to compute the register addresses, and only applies to the first source operand (src0) and/or the destination operand.

There is one address register that contains 8 sub-registers. Each sub-register contains a 16-bit unsigned value. The leading two sub-registers form a special doubleword that can be used as the descriptor for the send instruction.

Source operand can also be immediate value (also referred to as inline constants). For instructions with two source operands, only the second operand <src1> is allowed to be immediate. For instructions with only one source operand, the source operand <src0> is used and it can be an immediate.

An immediate source operand can be a scalar value of specified type up to 32-bit wide, which is replicated to create a vector with length of Execution Size. An immediate operand can also be a special 32-bit vector with 8 elements each of 4-bit signed integer value, or a 32-bit vector with 4 elements each of 8-bit restricted float value.

7.1.3 Instruction Execution

It is implied that all instructions operate across all channels of data unless otherwise specified either via destination mask, predication, execution mask (caused by SIMD branch and loop instructions), or execution size.

Instruction execution size can be specified per instruction, from scalar (*ExecSize* = 1) up to the maximal execution size supported for the data type, with the restriction that execution size can only be in power of 2.

7.2 Instruction Machine Formats

This section shows the machine formats of the instruction set. The instructions in the architecture have fixed length of 128 bits. Out of the 128 bits, there are 120 bits in use, and the remaining bits are reserved for future extensions. One instruction consists of instruction fields that control various stages of execution of the instruction. These fields are roughly groups into the 4 doublewords as the following.

- Instruction Operation Doubleword (DW0) contains the opcode and other general instruction control fields.
- Instruction Destination Doubleword (DW1) contains the destination operand (<dst>) and the register file and type of source operands.
- Instruction Source-0 Doubleword (DW2) contains the first source operand (<src0>) and flag register number
- Instruction Source-1 Doubleword (DW3) contains the second source operand (<src1>) and is used to hold the 32-bit immediate source (imm32 as <src0> or <src1>).

Figure 5-1. DevSNB Instruction Format – 1-src and 2-src depicts the details of the organization of fields in the 128-bit instruction word based on the Addressing Mode and Access Mode of an instruction. Definitions for individual instruction fields are provided in the following sections.

The ***send*** instruction is shown in the talbe as it has some unique instruction fields. For example, the message descriptor (plus EOT) occupies the whole DW3, and the immediate destination register overlaps with the Conditional Modifier field. The rest of fields in DW0-3 follows the definition on the left, depending on Addressing Mode and Access Mode of the ***send*** instruction.

The ***math*** and ***conditional branch*** instruction are also shown in the table as they have some unique instruction fields.

Not shown is for immediate operands. When an immediate source is present in an instruction, it always occupies the whole DW3 with a 32-bit value.

Support for indirect addressing for <src1>, as shown by the gray areas in Table 4-2 is device dependent. See Table 5-25 (Indirect source addressing support available in device hardware) in ISA Execution Environment for details.

Figure 5-1. DevSNB Instruction Format – 1-src and 2-src SNB

DW #	Instr Bits Alloc	High Bit	Low Bit	Instr Bits Used	AddrMode = Direct		AddrMode = Indirect		SEND		MATH	Conditional Branch	Break Continue	Imm Src
					AccessMode = Align16	AccessMode = Align1	AccessMode = Align16	AccessMode = Align1	MsgDesc Imm	MsgDesc Reg				
	1	127	127	1					EOT					
	2	126	125	2										
	4	124	121	4										
	4	120	117	4	Src1.VertStride									
	1	116	116	1										
	2	115	114	2		Src1.Width		Src1.Width						
	2	113	112	2	Src1.ChanSel[7:4]	Src1.HorzStride	Src1.ChanSel[7:4]	Src1.HorzStride					UIP[15:0]	
	1	111	111	1	Src1.AddrMode									
	2	110	109	2	Src1.SrcMod									
	3	108	106	3			Src1.AddrSubRegNum							
	5	105	101	5	Src1.RegNum [7:0]									
	1	100	100	1	Src1.SubRegNum [4]		Src1.AddrImm [9:4]							
3	4	99	96	4	Src1.ChanSel[3:0]	Src1.SubRegNum [4:0]	Src1.ChanSel[3:0]	Src1.AddrImm [9:0]	Imm[28:0]	Reg32	Same	Same	JIP[15:0]	Imm[31:0]
	6	95	90	0										
	1	89	89	1	FlagSubRegNum									
	4	88	85	4	Src0.VertStride									
	1	84	84	1										
	2	83	82	2		Src0.Width		Src0.Width						
	2	81	80	2	Src0.ChanSel[7:4]	Src0.HorzStride	Src0.ChanSel[7:4]	Src0.HorzStride						
	1	79	79	1	Src0.AddrMode									
	2	78	77	2	Src0.SrcMod									
	3	76	74	3			Src0.AddrSubRegNum							
	5	73	69	5	Src0.RegNum [7:0]									
	1	68	68	1	Src0.SubRegNum [4]		Src0.AddrImm [9:4]							
2	4	67	64	4	Src0.ChanSel[3:0]	Src0.SubRegNum [4:0]	Src0.ChanSel[3:0]	Src0.AddrImm [9:0]	Same		Same	Same	Same	Same
	1	63	63	1	Dst.AddrMode									
	2	62	61	2		Dst.HorzStride		Dst.HorzStride						
	3	60	58	3			Dst.AddrSubRegNum							
	5	57	53	5	Dst.RegNum [7:0]									
	1	52	52	1	Dst.SubRegNum [4]		Dst.AddrImm [9:4]							
	4	51	48	4	Dst.ChanEn[3:0]	Dst.SubRegNum [4:0]	Dst.ChanEn[3:0]	Dst.AddrImm [9:0]	Same		Same	JIP[15:0]	Same	Same
	1	47	47	0										
	3	46	44	3	Src1.SrcType									
	2	43	42	2	Src1.RegFile									
	3	41	39	3	Src0.SrcType									
	2	38	37	2	Src0.RegFile									
	3	36	34	3	Dst.DstType									
1	2	33	32	2	Dst.RegFile				Same		Same	Same	Same	Same
	1	31	31	1	Saturate									
	1	30	30	1	DebugCtrl									
	1	29	29	1	CmptCtrl									
	1	28	28	1	**AccWrCtrl**				Same		Same		Same	
	4	27	24	4	CondModifier				SFID[3:0]		FC[3:0]		MBZ	
	3	23	21	3	ExecSize									
	1	20	20	1	PredInv									
	4	19	16	4	PredCtrl				Same		Same			
	2	15	14	2	ThreadCtrl						FC[5:4]			
	2	13	12	2	QtrCtrl									
	2	11	10	2	DepCtrl									
	1	9	9	1	**WECtrl**									
	1	8	8	1	AccessMode				Same		Same	Same	Same	Same
	1	7	7	0	(reserved for Opcode)									
0	7	6	0	7	Opcode				Same		Same	Same	Same	Same

DevSNB also supports a separate instruction format for instructions with 3 sources. It havs fixed length of 128 bits. Out of the 128 bits, there are 114 bits in use, and the remaining bits are reserved for future extensions. One instruction consists of instruction fields that control various stages of execution of the instruction. These fields are roughly groups into the 4 doublewords as the following.

- Instruction Operation Doubleword (DW0) contains the opcode and other general instruction control fields.
- Instruction Destination Doubleword (DW1) contains the destination operand (<dst>), the register file and source modifiers, and flag subregister number.

- Instruction Source Doublewords (DW2/DW3) contains the three source operands (<src0>, <src1>, <src2>).

The 3-src instructions have the following restrictions compare to the 1-src/2-src instructions.

- The only supported instructions are: LRP, MAD, BFE, BFI2
- Only GRF register allowed for sources, and only GRF/MRF register allowed for destination
- Subregister number can only go down to DWord granularity.
- Must be Align16, uses Align16 style swizzling, with extra replication control. No other regioning support.

Figure 7-3 DevSNB Instruction Format – 3-src SNB

DW #	Instr Bits Alloc	High Bit	Low Bit	Instr Bits Used	Description
	2	127	126	0	*reserved*
	8	125	118	8	Src2 Regnum
	3	117	115	3	Src2 Subregnum
	8	114	107	8	Src2 Swizzle
	1	106	106	1	Src2 RepCtrl
	1	105	105	0	*reserved*
	8	104	97	8	Src1 Regnum
	3	96	94	3	Src1 Subregnum
	8	93	86	8	Src1 Swizzle
	1	85	85	1	Src1 RepCtrl
	1	84	84	0	*reserved*
	8	83	76	8	Src0 Regnum
	3	75	73	3	Src0 Subregnum
	8	72	65	8	Src0 Swizzle
2,3	1	64	64	1	Src0 RepCtrl
	8	63	56	8	Dst Regnum
	3	55	53	3	Dst Subregnum
	4	52	49	4	Dst chan enable
	7	48	42	0	*reserved*
	2	41	40	2	Src2 Modifier
	2	39	38	2	Src1 Modifier
	2	37	36	2	Src0 Modifier
	2	35	34	0	*reserved*
	1	33	33	1	FlagSubRegNum
1	1	32	32	1	Dst Regfile
	1	31	31	1	*Saturate*
	1	30	30	1	*DebugCtrl*
	1	29	29	1	*CmptCtrl*
	1	28	28	1	***AccWrCtrl***
	4	27	24	4	*CondModifier*
	3	23	21	3	*ExecSize*
	1	20	20	1	*PredInv*
	4	19	16	4	*PredCtrl*
	2	15	14	2	*ThreadCtrl*
	2	13	12	2	*QtrCtrl*
	2	11	10	2	*DepCtrl*
	1	9	9	1	***WECtrl***
	1	8	8	1	*AccessMode*
	1	7	7	0	*(reserved for Opcode)*
0	7	6	0	7	*Opcode*

7.2.1 Common Instruction Fields

7.2.1.1 1-src and 2-src Instructions

As shown in the table below, the meanings (encoding) of certain bit fields in the 128-bit instruction word varies depending on the values of other bit fields.

Table 7-2 provides the definition of common fields in the instruction word. The 'Width' column specifies the width of the field in bits. These common fields will be referred to later in describing the fields of different doublewords of the instruction. The definition for fields that have unique representation can be found in its corresponding doubleword of the instruction.

Table 7-2. Definitions of Common Instruction Fields

Field	Description	Width
CondModifier	**Conditional Modifier.** This field sets the flag register based on the internal conditional signals output from the execution pipe such as sign, zero, overflow and NaNs, etc. If this field is set to 0000, no flag registers are updated. Flag registers are not updated for instructions with embedded compares. This field may also be referred to as the *flag destination control* field. This field applies to all instructions except *send, sendc, and math*. 0000 = Do not modify the flag register (normal) 0001 = Zero or Equal ('**.z**' or '**.e**') 0010 = Not Zero or Not Equal ('**.nz**' or '**.ne**') 0011 = Greater-than ('**.g**') 0100 = Greater-than-or-equal ('**.ge**') 0101 = Less-than ('**.l**') 0110 = Less-than-or-equal ('**.le**') 0111 = Reserved 1000 = Overflow ('**.o**') 1001 = Unordered with Computed NaN ('**.u**') 1010 -1111 = Reserved	4
AddrMode	**Addressing Mode.** This field determines the addressing method of the operand. When it is cleared, the register address of the operand is directly provided by bits in the instruction word. It is called a direct register addressing mode. When it is set, the register address of the operand is computed based on the address register value and an address immediate field in the instruction word. This is referred to as a register-indirect register addressing mode. This field applies to the destination operand and the first source operand, <src0>. Support for <src1> is device dependent. See Table XX (Indirect source addressing support available in device hardware) in ISA Execution Environment for details. 0 = "Direct". Direct register addressing 1 = "Register-Indirect" (or in short "Indirect"). Register-indirect register addressing	1

Field	Description	Width
RegNum	**Register Number.** This field provides the register number for the operand. For GRF or MRF register operand, it provides the portion of register address aligning to 256-bit. For an ARF register operand, this field is encoded such that MSBs identify the architecture register type and LSBs provide its register number. This field together with the corresponding *SubRegNum* field provides the byte aligned address for the origin of the register region. Specifically, this field provides bits [12:5] of the byte address, while *SubRegNum* field provides bits [4:0]. This field applies to the destination operand and the source operands. It is ignored (or not present in the instruction word) for an immediate source operand. This field is present if the operand is in direct addressing mode; it is not present if the operand is register-indirect addressed. Format = U8, if *RegFile* = **GRF**. 0x00 to 0x7F = Register number in the range of [0, 127] 0x80 to 0xFF = Reserved Format = U8, if *RegFile* = **MRF**. 0x00 to 0x0F = Register number in the range of [0, 15] 0x10 to 0xFF = Reserved Format = 8-bit encoding, if *RegFile* = **ARF**. This field is used to encode the architecture register as well as providing the register number. See Execution Environment chapter for details.	8
SubRegNum	**Sub-Register Number.** This field provides the sub-register number for the operand. For GRF or MRF register operand, it provides the byte address within a 256-bit register. For an ARF register operand, this field also provides the sub-register number according to special encoding for the given architecture register. This field together with the corresponding *RegNum* field provides the byte aligned address for the origin of the register region. Specifically, this field provides bits [4:0] of the byte address, while *RegNum* field provides bits [12:5]. This field applies to the destination operand and the source operands. It is ignored (or not present in the instruction word) for an immediate source operand. This field is present if the operand is in direct addressing mode; it is not present if the operand is register-indirect addressed. Format = U5, if *RegFile* = **GRF** or **MRF** 0x00 to 0x1F = Sub-Register number in the range of [0, 31] Format = 5-bit encoding, if *RegFile* = **ARF**. This field is used to encode the architecture register as well as providing the register number. See Execution Environment chapter for details.	5
AddrSubRegNum	**Address Sub-Register Number.** This field provides the sub-register number for the address register. The address register contains 8 sub-registers. The size of each sub-register is one word. The address register contains the register address of the operand, when the operand is in register-indirect addressing mode. This field applies to the destination operand and the source operands. It is ignored (or not present in the instruction word) for an immediate source operand. This field is present if the operand is in register-indirect addressing mode; it is not present if the operand is directly addressed. Format = U3 0x00 to 0x07 = Address Sub-Register number in the range of [0, 7]	3

Field	Description	Width
AddrImm	**Address Immediate.** This field provides the immediate value in unit of byte to be added to the address register in order to compute the register address (byte-aligned region origin) for the operand. It is a 10-bit signed integer in 2's compliment form. This field is present if the operand is in register-indirect addressing mode; it is not present if the operand is directly addressed. *Note: that the address immediate field may not be able to cover the whole GRF register range for a thread, as the maximum GRF register space for a thread is 4KB.* Format = S9 Valid range: [-512, 511]	10
SrcMod	**Source Modifier.** This field specifies the numerical modification to a source operand. The value of each data element of a source operand can optionally have its absolute value taken and/or its sign inverted prior to delivery to the execution pipe. The absolute value is prior to negate such that a guaranteed negative value can be produced. This field only applies to source operand. It does not apply to destination. This field is not present for an immediate source operand. 00 = No modification (normal) 01 = **"(abs)"**. Absolute 10 = **"–"**. Negate 11 = **"–(abs)"**. Negate of the absolute (forced negative value)	2

Field	Description	Width
VertStride	**Vertical Stride.** The field provides the vertical stride of the register region in unit of data elements for an operand. Encoding of this field provides values in power of 2, ranging from 0 to 32 elements. Larger values are not supported due to the restriction that a source operand must reside within two adjacent 256-bit registers (64 bytes total). Special encoding 1111b (0xF) is only valid when the operand is in register-indirect addressing mode (*AddrMode* = 1). If this field is set to 0xF, one or more sub-registers of the address registers may be used to compute the addresses. Each address sub-register provides the origin for a row of data element. The number of address sub-registers used is determined by the division of *ExecSize* of the instruction by the *Width* fields of the operand. This field only applies to source operand. It does not apply to destination. This field is not present for an immediate source operand. For **Align16** access mode, only encodings of 0000 and 0011 are allowed. Other codes are reserved. *Note 1: Vertical Stride larger than 32 is not allowed due to the restriction that a source operand must reside within two adjacent 256-bit registers (64 bytes total).* *Note 2: In **Align16** access mode, as encoding 0xF is reserved, only single-index indirect addressing is supported.* *Note 3: If indirect address is supported for <src1>, encoding 0xF is reserved for <src1> – only single-index indirect addressing is supported.* 0000 = 0 Elements 0001 = 1 Element 0010 = 2 Elements 0011 = 4 Elements 0100 = 8 Elements 0101 = 16 Elements (applies to byte or word operand only) 0110 = 32 Elements (applies to byte operand only) 0111-1110 = Reserved 1111 = **VxH or Vx1 mode** (only valid for register-indirect addressing in **Align1** mode)	4
Width	**Width.** This field specifies the number of elements in the horizontal dimension of the region for a source operand. This field cannot exceed the *ExecSize* field of the instruction. This field only applies to source operand. It does not apply to destination. This field is not present for an immediate source operand. 000 = 1 Elements 001 = 2 Elements 010 = 4 Elements 011 = 8 Elements 100 = 16 Elements 101-111 = Reserved	3

Field	Description	Width
HorzStride	**Horizontal Stride.** This field provides the distance in unit of data elements between two adjacent data elements within a row (horizontal) in the register region for the operand. This field applies to both destination and source operands. This field is not present for an immediate source operand. 00 = 0 Elements 01 = 1 Element 10 = 2 Elements 11 = 4 Elements	2
Imm32	**32-bit Immediate.** The 32-bit immediate data field for the operand. It may contain any legal bit pattern for its associated type. Only one 32-bit immediate value may be present in an instruction, therefore binary operations only support <src1> as an immediate value. The low order bits are directly used when fewer than 32-bits are needed to describe the desired type; the 32-bits are not coerced into the designated type. For UW and W data types, programmer is required to replicate the lower word to the upper word of this field. This field only applies to the last source operand. Signed and unsigned byte integer data types are not supported for an immediate operand. Valid ranges according to data type: **Immediate Data Type** / **Valid Range (inclusive)** F: [0…±1.0*$2^{-128...127}$] UW: [0, 65535] W: [-32768, 32767] UD: [0, 2^{32}-1] D: [-2^{31}, 2^{31}-1] VF: [0, ±0.125…±31] V: [-8, 7]	32
ChanEn	**Channel Enable.** Four channel enables are defined for controlling which channels will be written into the destination region. These channel mask bits are applied in a modulo-four manner to all *ExecSize* channels. There is 1-bit Channel Enable for each channel within the group of 4. If the bit is cleared, the write for the corresponding channel is disabled. If the bit is set, the write is enabled. Mnemonic for the bit being set for the group of 4 is “x”, “y”, “z”, and “w”, respectively, where “x” corresponds to Channel 0 in the group and “w” corresponds to channel 3 in the group. This field only applies to destination operand. This field is only present in **Align16** mode. 0 = Write Disabled 1 = Write Enabled (normal)	4

Field	Description	Width
ChanSel	**Channel Select.** This field controls the channel swizzle for a source operand. The normally sequential channel assignment can be altered by explicitly identifying neighboring data elements for each channel. Out of the 8-bit field, 2 bits are assigned for each channel within the group of 4. ChanSel[1:0], [3.2], [5.4] and [7,6] are for channel 0 ("x"), 1 ("y"), 2 ("z"), and 3 ("w") in the group, respectively. For example with an execution size of 8, *r0.0**<4>.zywz:f*** would assign the channels as follows: $Chan_0 = Data_2$, $Chan_1 = Data_1$, $Chan_2 = Data_3$, $Chan_3 = Data_2$; $Chan_4 = Data_6$, $Chan_5 = Data_5$, $Chan_6 = Data_7$, $Chan_7 = Data_6$. This field only applies to source operand. This field is only present in **Align16** mode. It is not present for an immediate source operand. The 2-bit Channel Selection field for each channel within the group of 4 is defined as the following. 00 = **"x"**. Channel 0 is selected for the corresponding execution channel 01 = **"y"**. Channel 1 is selected for the corresponding execution channel 10 = **"z"**. Channel 2 is selected for the corresponding execution channel 11 = **"w"**. Channel 3 is selected for the corresponding execution channel	8
MsgDscpt31	**Message Description.** This field, containing 31-bit immediate values, provides the description of the message to be sent. This field only applies to the *send* instruction. It is not present for other instructions. The meaning of the field depends on the type of message as well as the message shared function target. Format: U31	31
EOT	**End of Thread.** This field controls the termination of the thread. For a *send* instruction, if this field is set, EU will terminate the thread and also set the EOT bit in the message sideband. This field only applies to the *send* instruction. It is not present for other instructions. 0 = The thread is not terminated 1 = **EOT**	1

7.2.1.2 3-src Instructions

The table in this section describes the encoding for the common fields for the 3-src instructions format.

Table 7-3. Definitions of Common Instruction Fields

Field	Description	Width
CondModifier	**Conditional Modifier.** This field sets the flag register based on the internal conditional signals output from the execution pipe such as sign, zero, overflow and NaNs, etc. If this field is set to 0000, no flag registers are updated. Flag registers are not updated for instructions with embedded compares. This field may also be referred to as the *flag destination control* field. This field applies to all instructions except *send, sendc, and math*. 0000 = Do not modify the flag register (normal) 0001 = Zero or Equal ('**.z**' or '**.e**') 0010 = Not Zero or Not Equal ('**.nz**' or '**.ne**') 0011 = Greater-than ('**.g**') 0100 = Greater-than-or-equal ('**.ge**') 0101 = Less-than ('**.l**') 0110 = Less-than-or-equal ('**.le**') 0111 = Reserved 1000 = Overflow ('**.o**') 1001 = Unordered with Computed NaN ('**.u**') 1010 -1111 = Reserved	4
RegNum	**Register Number.** This field provides the register number for the operand. For GRF or MRF register operand, it provides the portion of register address aligning to 256-bit. For an ARF register operand, this field is encoded such that MSBs identify the architecture register type and LSBs provide its register number. This field together with the corresponding *SubRegNum* field provides the byte aligned address for the origin of the register region. Specifically, this field provides bits [12:5] of the byte address, while *SubRegNum* field provides bits [4:0]. This field applies to the destination operand and the source operands. It is ignored (or not present in the instruction word) for an immediate source operand. This field is present if the operand is in direct addressing mode; it is not present if the operand is register-indirect addressed. Format = U8, if *RegFile* = **GRF**. 0x00 to 0x7F = Register number in the range of [0, 127] 0x80 to 0xFF = Reserved Format = U8, if *RegFile* = **MRF**. 0x00 to 0x0F = Register number in the range of [0, 15] 0x10 to 0xFF = Reserved	8
SubRegNum	**Sub-Register Number.** This field provides the sub-register number for the operand, it provides the **dword** address within a 256-bit register This field together with the corresponding *RegNum* field provides the dword aligned address for the origin of the register region. Specifically, this field provides bits [4:2] of the dword address, while *RegNum* field provides bits [12:5]. This field applies to the destination operand and the source operands.	3

Field	Description	Width
SrcMod	**Source Modifier.** This field specifies the numerical modification to a source operand. The value of each data element of a source operand can optionally have its absolute value taken and/or its sign inverted prior to delivery to the execution pipe. The absolute value is prior to negate such that a guaranteed negative value can be produced. This field only applies to source operand. It does not apply to destination. This field is not present for an immediate source operand. 00 = No modification (normal) 01 = **"(abs)"**. Absolute 10 = **"–"**. Negate 11 = **"–(abs)"**. Negate of the absolute (forced negative value)	2
ChanSel	**Channel Select.** This field controls the channel swizzle for a source operand. The normally sequential channel assignment can be altered by explicitly identifying neighboring data elements for each channel. Out of the 8-bit field, 2 bits are assigned for each channel within the group of 4. ChanSel[1:0], [3.2], [5.4] and [7,6] are for channel 0 ("x"), 1 ("y"), 2 ("z"), and 3 ("w") in the group, respectively. For example with an execution size of 8, *r0.0**<4>.zywz:f*** would assign the channels as follows: $Chan_0 = Data_2$, $Chan_1 = Data_1$, $Chan_2 = Data_3$, $Chan_3 = Data_2$; $Chan_4 = Data_6$, $Chan_5 = Data_5$, $Chan_6 = Data_7$, $Chan_7 = Data_6$. This field only applies to source operand. This field is only present in **Align16** mode. It is not present for an immediate source operand. The 2-bit Channel Selection field for each channel within the group of 4 is defined as the following. 00 = **"x"**. Channel 0 is selected for the corresponding execution channel 01 = **"y"**. Channel 1 is selected for the corresponding execution channel 10 = **"z"**. Channel 2 is selected for the corresponding execution channel 11 = **"w"**. Channel 3 is selected for the corresponding execution channel	8
RepCtrl	**Replicate Control.** This field controls the replication of the starting channel to all channels in the execution size. This field applies to all three source operands. 0 = No replication 1 = Replicate across all channels	1

7.2.2 Instruction Operation Doubleword (DW0)

Most fields in Instruction Operation Doubleword (DW0) apply to all instructions. Bit field [27:24] is one exception. It is *CondModifier* for most instructions but is *CurrDest.RegNum* field for the *send* instruction.

The descriptions in the table below are shared between the 1-src/2-src instructions and 3-src instructions.

Table 7-4. Definitions of Fields in Operation Doubleword (DW0)

<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>31</td><td>
Saturate. This field controls the destination saturation.
When it is set, output data to the destination register are saturated. The saturation operation depends on the destination data type. Saturation is the operation that converts any data that is outside the saturation target range for the data type to the closest represented value with the target range. If destination type is float, saturation target range is [0, 1]. For example, any positive number greater than 1 (including +INF) is saturated to 1 and any negative number (including –INF) is saturated to 0. A NaN is saturated to 0, For integer data types, the maximum range for the given numerical data type is the saturation target range.
When it is not set, output data to the destination register are not saturated. For example, a wrapped result (modular) is output to the destination for an overflowed integer data.
More details can be found in the Data Types chapter.
0 = No destination modification (normal)
1 = “sat”. Saturate the output
<table>
<tr><th>Destination Type</th><th>Saturation Target Range (inclusive)</th></tr>
<tr><td>Float (F)</td><td>[0.0, 1.0]</td></tr>
<tr><td>Byte (UB)</td><td>[0, 255]</td></tr>
<tr><td>Signed Byte (B)</td><td>[-128, 127]</td></tr>
<tr><td>Word (UW)</td><td>[0, 65535]</td></tr>
<tr><td>Signed Word (W)</td><td>[-32768, 32767]</td></tr>
<tr><td>Double Word (UD)</td><td>[0, 2^{32}-1]</td></tr>
<tr><td>Signed Double (D)</td><td>[-2^{31}, 2^{31}-1]</td></tr>
</table>
</td></tr>
<tr><td>30</td><td>Reserved</td></tr>
<tr><td>29</td><td>Reserved: MBZ</td></tr>
<tr><td>28</td><td>
AccWrCtrl. This field allows per instruction accumulator write control.
0 = don’t write result into accumulator
1 = “AccWrCtrl”. write result into accumulator, and destination
</td></tr>
<tr><td>27:24</td><td>
CondModifier or CurrDst.RegNum[3:0]
Definition of this bit field depends on whether the instruction is a send/math or not.
<table>
<tr><th>Opcode != ‘send’</th><th>Opcode = ‘send’</th></tr>
<tr><td>CondModifier:
This field sets the flag register based on the internal conditional signals output from the execution pipe.</td><td>CurrDst.RegNum[3:0]
This field sets the MRF register number for the current destination operand in the send instruction. No flag registers are updated for the send instruction. The 4-bit field provides full access of the 16 MRF registers.
(See Instruction Reference chapter for CurrDst.)</td></tr>
</table>
</td></tr>
</table>

Bits	Description
23:21	**ExecSize – Execution Size.** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type. 000 = 1 Channels (scalar operation) 001 = 2 Channels 010 = 4 Channels 011 = 8 Channels 100 = 16 Channels 101= 32 Channels 110-111 = Reserved
20	**PredInv – Predicate Inverse.** This field, together with *PredCtrl*, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 – there is no predication. 0 = **"+"**. Positive polarity of predication. 1 = **"–"**. Negative polarity of predication.
19:16	**PredCtrl – Predicate Control.** This field, together with *PredInv*, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register. Encoding depends on the access mode. In **Align16** access mode, there are eight encodings (including no predication). All encodings are based on group-of-4 predicate bits, including channel sequential, replication swizzles and horizontal any|all operations. The same configuration is repeated for each group-of-4 execution channels. In **Align1** access mode, there are twelve encodings (including no predication). The encodings applies to all execution channels with explicit channel grouping from single channel up to group of 16 channels. Predicate Control in **Align16** access mode 0000 = No predication (normal) 0001 = Predication with sequential flag channel mapping 0010 = Predication with replication swizzle **'.x'** 0011 = Predication with replication swizzle **'.y'** 0100 = Predication with replication swizzle **'.z'** 0101 = Predication with replication swizzle **'.w'** 0110 = Predication with **'.any4h'** 0111 = Predication with **'.all4h'** 1000 -1111 = Reserved Predicate Control in **Align1** access mode 0000 = No predication (normal) 0001 = Predication with sequential flag channel mapping 0010 = Predication with **.anyv** (any from f0.0-f0.1 on the same channel) 0011 = Predication with **.allv** (all of f0.0-f0.1 on the same channel) 0100 = Predication with **.any2h** (any in group of 2 channels) 0101 = Predication with **.all2h** (all in group of 2 channels) 0110 = Predication with **.any4h** (any in group of 4 channels) 0111 = Predication with **.all4h** (all in group of 4 channels) 1000 = Predication with **.any8h** (any in group of 8 channels) 1001 = Predication with **.all8h** (all in group of 8 channels)

Bits	Description
	1010 = Predication with **.any16h** (any in group of 16 channels) 1011 = Predication with **.all16h** (all in group of 16 channels) 1100 = Reserved 1101 = Reserved 1100 -1111 = Reserved **[DevSNB]**
15:14	**ThreadCtrl – Thread Control.** This field provides explicit control for thread switching. If this field is set to 00, it is up to the execution units to manage thread switching. This is the normal operations mode. In this mode, for example, if the current instruction cannot proceed due to operand dependencies, EU switches to next available thread to fill the compute pipe. In another example, if the current instruction is ready to go, however, there is another thread with higher priority also has instruction ready, EU switches to that thread. If this field is set to **Switch**, a forced thread switch occurs after the current instruction is executed and before the next instruction. In addition, a long delay (longer than the execution pipe latency) for the current thread is introduced for the thread. Particularly, the instruction queue of the current thread is flushed after the current instruction is dispatched for execution. If this field is set to **Atomic**, the next instruction will get highest priority in the thread arbitration for the exeuction pipelines. **Switch** is designed primarily as a safety feature in case there are race conditions for certain instructions. 00 = Normal Thread Control 10 = **"Switch"** 01 = **"Atomic"** 11 = Reserved
13:12	**QtrCtrl – Quarter Control.** This field provides explicit control for ARF selection. This filed combines with **ExecSize** determines which channels are used for the ARF registers.

QtrCtrl	NibCtrl	ExecSize	Description	BNF
00	x	8	use first quarter for DMask/VMask use first half for everything else	**1Q**
01	x	8	use second quarter for DMask/VMask use second half for everything else	**2Q**
10	x	8	use third quarter for DMask/VMask use first half for everything else	**3Q**
11	x	8	use forth quarter for DMask/VMask use second half for everything else	**4Q**
0x	x	16	use first half for DMask/VMask use all channels for everything else	**1H**
1x	x	16	use second half for DMask/VMask use all channels for everything else	**2H**
00	0	4	use first 1/8 for DMask/VMask and ARF	**1N**
00	1	4	use second 1/8 for DMask/VMask and ARF	**2N**
01	0	4	use thrid 1/8 for DMask/VMask and ARF	**3N**
01	1	4	use forth 1/8 for DMask/VMask and ARF	**4N**
10	0	4	use fifth 1/8 for DMask/VMask and ARF	**5N**

<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><td>10</td><td>1</td><td>4</td><td>use sixth 1/8 for DMask/VMask and ARF</td><td>6N</td></tr>
<tr><td>11</td><td>0</td><td>4</td><td>use seventh 1/8 for DMask/VMask and ARF</td><td>7N</td></tr>
<tr><td>11</td><td>1</td><td>4</td><td>use eighth 1/8 for DMask/VMask and ARF</td><td>8N</td></tr>
</table>
2H is only allowed for SIMD16 instruction in Single Program Flow mode (SPF=1).
NibCtrl is only allowed for SIMD4 instructions with (DF) double precision source and/or destination.
</td></tr>
<tr><td>11:10</td><td>
DepCtrl – Destination Dependency Control. This field selectively disables destination dependency check and clear for this instruction.
When it is set to 00, normal destination dependency control is performed for the instruction – hardware checks for destination hazards to ensure data integrity. Specifically, destination register dependency check is conducted before the instruction is made ready for execution. After the instruction is executed, the destination register scoreboard will be cleared when the destination operands retire.
When bit 10 is set (NoDDClr), the destination register scoreboard will NOT be cleared when the destination operands retire. When bit 11 is set (NoDDChk), hardware does not check for destination register dependency before the instruction is made ready for execution. NoDDClr and NoDDChk are not mutual exclusive.
When this field is not all-zero, hardware does not protect against destination hazards for the instruction. This is typically used to assemble data in a fine grained fashion (e.g. matrix-vector compute with dot-product instructions), where the data integrity is guaranteed by software based on the intended usage of instruction sequences.
00 = Destination dependency checked and cleared (normal)
01 = “NoDDClr”. Destination dependency checked but not cleared
10 = “NoDDChk”. Destination dependency not checked but cleared
11 = “NoDDClr, NoDDChk”. Destination dependency not checked and not cleared
[Errata DevSNB] DepCtrl is not supported for ARF registers.
</td></tr>
<tr><td>9</td><td>
WECtrl – Write Enable Control. This field determines if the the per channel write enables are used to generate the final write enable. This field should be normally “0”.
0 = use normal write enables (normal)
1 = write all channels, except channels killed with predication control
</td></tr>
<tr><td>8</td><td>
AccessMode – Access Mode. This field determines the operand access for the instruction. It applies to all source and destination operands.
When it is cleared (Align1), the instruction uses byte-aligned addressing for source and destination operands. Source swizzle control and destination mask control are not supported.
When it is set (Align16), the instruction uses 16-byte-aligned addressing for all source and destination operands. Source swizzle control and destination mask control are supported in this mode.
0 = “Align1”
1 = “Align16”
</td></tr>
<tr><td>7</td><td>Reserved: MBZ (for future opcode extension)</td></tr>
<tr><td>6:0</td><td>
Opcode – Instruction Operation Code. This field contains the instruction operation code. Each opcode is given a unique mnemonic. For example, opcode 0x01 is for a move operation. Mnemonic for this opcode is mov.
See Section 7.3 for details of opcode encoding.
</td></tr>
</table>

7.2.3 Instruction Destination Doubleword (DW1)

7.2.3.1 1-src and 2-src Instructions

Destination Doubleword (DW1) contains the register file and numeric type of all operands, as well as the register region parameters of the destination operand. Table 7-5 shows the field definition of the Instruction Destination Doubleword. Furthermore, the Destination Register Region is described in Table 7-6 through Table 7-9.

Table 7-5. Instruction Destination Doubleword

Bits	Description
31:16	**Destination Register Region.** This word contains the parameters describing the register region of the destination operand. Subfield definition depends on the AccessMode. Detailed descriptions can be found in Table 7-6 through Table 7-9. **Programming Notes:** Allthough *Dst.HorzStride* is a don't care for Align16, HW needs this to be programmed as "01".
15	Reserved: MBZ
14:12	**Src1.SrcType – Source-1 Data Type.** This field specifies the numerical data type of the source operand <src1>. The bits of a source operand are interpreted as the identified numerical data type, rather than coerced into a type implied by the operator. Depending on *RegFile* field of the source operand, there are two different encoding for this field. If a source is a register operand, this field follows the Source Register Type Encoding. If a source is an immediate operand, this field follows the Source Immediate Type Encoding. Source Register Type Encoding is identical to that for Destination Type. Source Immediate Type Encoding differs in two areas. First, it does not support byte and unsigned numerical data types. Secondly, it has two 32-bit vector types – halfbyte integer vector (V) type and exponent-only float vector (VF) type. *Implementation Note 1: Both source operands, <src0> and <src1>, support immediate types, but only one immediate is allowed for a given instruction and it must be the last operand.* *Implementation Note 2: Halfbyte integer vector (v) type can only be used in instructions in packed-word execution mode. Therefore, in a two-source instruction where <src1> is of type :v, <src0> must be of type :b, :ub, :w, or :uw.* Source Register Type Encoding 000 = **"UD"**. **U**nsigned **D**oubleword integer 001 = **"D"**. Signed **D**oubleword integer 010 = **"UW"**. **U**nsigned **W**ord integer 011 = **"W"**. Signed **W**ord integer 100 = **"UB"**. **U**nsigned **B**yte integer 101 = **"B"**. Signed **B**yte integer 110 = Reserved **[DevSNB]** 111 = **"F"**. Single precision **F**loat (32-bit) Source Immediate Type Encoding: 000 = **"UD"** 001 = **"D"** 010 = **"UW"** 011 = **"W"**

Bits	Description
	100 = "UV". 32-bit halfbyte **U**nsigned Integer **V**ector 101 = **"VF"**. 32-bit restricted **V**ector **F**loat 110 = **"V"**. 32-bit halfbyte integer **V**ector 111 = **"F"**
11:10	**Src1.RegFile – Source-1 Register File.** This field identifies the register file of source operand <src1>. 00 = **"ARF"**. Architecture Register File (**a#**, **acc#**, **f#**, **n#**, **null**, **ip**, etc.) 01 = **"GRF"**. General Register File (**r#**) 10 = **"MRF"**. Message Register File (**m#**) 11 = **"IMM"**. Immediate
9:7	**Src0.SrcType – Source-0 Data Type.** This field is the *SrcType* for <src0> operand. It has the same definitions as *Src1.SrcType*.
6:5	**Src0.RegFile – Source-0 Register File.** This field is the *RegFile* for <src0> operand. It has the same definitions as *Src1.RegFile*.
4:2	**Dst.DstType – Destination Data Type.** This field specifies the numerical data type of the destination operand <dst>. The bits of the destination operand are interpreted as the identified numerical data type, rather than coerced into a type implied by the operator. For a *send* instruction, this field applies to the **CurrDst** – the current destination operand. Encoding: 000 = **"UD"**. **U**nsigned **D**oubleword integer 001 = **"D"**. Signed **D**oubleword integer 010 = **"UW"**. **U**nsigned **W**ord integer 011 = **"W"**. Signed **W**ord integer 100 = **"UB"**. **U**nsigned **B**yte integer 101 = **"B"**. Signed **B**yte integer 110 = Reserved 111 = **"F"**. Single precision **F**loat (32-bit)
1:0	**Dst.RegFile – Destination Register File.** This field identifies the register file of the destination operand <dst>. Note that it is obvious that immediate cannot be a destination operand. For a *send* instruction, this field applies to the **PostDst** – the post destination operand. Encoding: 00 = **"ARF"**. Architecture Register File (**a#**, **acc#**, **f#**, **n#**, **null**, **ip**, etc.) 01 = **"GRF"**. General Register File (**r#**) 10 = **"MRF"**. Message Register File (**m#**) 11 = **reserved**

The following tables describe the Destination Register Region based on the access mode and addressing mode.

Table 7-6. Destination Register Region in Direct + Align16 mode

Bits	Description
15	**Dst.AddrMode – Destination Address Mode.** This field is the *AddrMode* for the destination operand. (See section 0 for definition of *AddrMode.*) For a *send* instruction, this field applies to **PostDst** – the post destination operand. Addressing mode for *CurrDst* (current destination operand) is fixed as Direct. (See Instruction Reference chapter for *CurrDst* and *PostDst*.)
14:13	Reserved: MBZ
12:5	**Dst.RegNum – Destination Register Number.** This field is the *RegNum* field for the destination operand. (See section 0 for definitions of *RegNum.*) For a *send* instruction, this field applies to **PostDst**.
4	**Dst.SubRegNum[4].** This is the 16-byte aligned sub-register address. (See section 0 for definitions of *SubRegNum*) For a *send* instruction, this field applies to **CurrDst**.
3:0	**Dst.ChanEn – Destination Channel Enable**. The channel enable field for the destination operand. (See section 0 for definitions of *ChanEn*) For a *send* instruction, this field applies to the **CurrDst**.

Table 7-7. Destination Register Region in Direct+Align1 mode

Bits	Description
15	**Dst.AddrMode – Destination Address Mode.** This field is the *AddrMode* for the destination operand. For a *send* instruction, it applies to **PostDst**. Addressing mode for ***CurrDst*** is fixed as Direct.
14:13	**Dst.HorzStride – Destination Horizontal Stride.** This field is the *HorzStride* for the destination operand. For a *send* instruction, this field applies to **CurrDst**. **PostDst** only uses the register number.
12:5	**Dst.RegNum – Destination Register Number.** This field is the *RegNum* field for the destination operand. For a *send* instruction, this field applies to **PostDst**.
4:0	**Dst.SubRegNum – Destination Sub-Register Number.** This field is the **SubRegNum** for the destination operand. (See section 0 for definition of *SubRegNum*) For a *send* instruction, this field applies to **CurrDst**.

Table 7-8. Destination Register Region in Indirect+Align16 mode

Bits	Description
15	**Dst.AddrMode – Destination Address Mode.** This field is the *AddrMode* for the destination operand. For a *send* instruction, this field applies to **PostDst**. Addressing mode for ***CurrDst*** is fixed as Direct.
14:13	Reserved: MBZ
12:10	**Dst.AddrSubRegNum – Destination Address Sub-Register Number.** This field is the *AddrSubRegNum* for the destination operand. (See section 0 for definition of *AddrSubRegNum.*) For a *send* instruction, this field applies to **PostDst**.

Bits	Description
9:4	**Dst.AddrImm[9:4]** This is the half-register aligned *AddrImm* field for the destination operand. (See section 0 for definition of *AddrImm*) For a *send* instruction, this field applies to **PostDst**.
3:0	**Dst.ChanEn – Destination Channel Enable.** The channel enable field for the destination operand. For a *send* instruction, this field applies to the **CurrDst**.

Table 7-9. Destination Register Region in Indirect+Align1 mode

Bits	Description
15	**Dst.AddrMode – Destination Address Mode.** This field is the *AddrMode* for the destination operand. For a *send* instruction, this field applies to **PostDst**. Addressing mode for ***CurrDst*** is fixed as Direct.
14:13	**Dst.HorzStride – Destination Horizontal Stride** This field is the *HorzStride* for the destination operand. For a *send* instruction, this field applies to **CurrDst**. **PostDst** only uses the register number.
12:10	**Dst.AddrSubRegNum – Destination Address Sub-Register Number.** This field is the *AddrSubRegNum* for the destination operand. For a *send* instruction, this field applies to **PostDst**.
9:0	**Dst.AddrImm – Destination Address Immediate.** This field is the byte-aligned *AddrImm* for the destination operand. For a *send* instruction, this field applies to **PostDst**.

7.2.3.2 3-src Instructions

This section describes the field in DW1 of the 3-src instruction format.

Table 7-10. Instruction DW1

Bits	Description
31:24	**Destination Register Number.** This field contains the destination register number.
23:21	**Destination Subregister Number.** This field contains the destination subregister number.
20:17	**Destination Channel Enable.** Four channel enables are defined for controlling which channels will be written into the destination region. These channel mask bits are applied in a modulo-four manner to all *ExecSize* channels. There is 1-bit Channel Enable for each channel within the group of 4. If the bit is cleared, the write for the corresponding channel is disabled. If the bit is set, the write is enabled. Mnemonic for the bit being set for the group of 4 is “x”, “y”, “z”, and “w”, respectively, where “x” corresponds to Channel 0 in the group and “w” corresponds to channel 3 in the group. 0 = Write Disabled 1 = Write Enabled (normal)
16:15	Reserved: MBZ
14:13	Reserved: MBZ
12:10	Reserved: MBZ

Bits	Description
9:8	**Source2 Modifier.** This field contains the modifier for source2. Refer to Table. 5-5 for the encoding.
7:6	**Source1 Modifier.** This field contains the modifier for source1. Refer to Table. 5-5 for the encoding.
5:4	**Source0 Modifier.** This field contains the modifier for source0. Refer to Table. 5-5 for the encoding.
3	Reserved: MBZ
2	Reserved: MBZ
1	**Flag Subregister Number.** This field contains the flag subregister number for instructions with non-zero Conditional Modifier.
0	**Destination Register File.** This filed contains the register file for destination. Only GRF and MRF are valid. 0 = GRF 1 = MRF

7.2.4 Instruction Source-0 Doubleword (DW2)

7.2.4.1 1-src and 2-src Instructions

Instruction Source-0 Doubleword (DW2) contains the first source operand and also flag register number.

- Table 7-11 shows the field definition for Direct Addressing with Align16.
- Table 7-12 shows the field definition for Direct Addressing with Align1.
- Table 7-13 shows the field definition for Indirect Addressing with Align16.
-

Table 7-14 shows the field definition for Indirect Addressing with Align1.

Table 7-11. Instruction Source-0 Doubleword in Direct+Align16 mode

Bits	Description
31:26	Reserved: MBZ
25	**FlagSubRegNum – Flag Sub-Register Number.** This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.
24:21	**Src0.VertStride – Source-0 Vertical Stride.** This field is the *VertStride* for <src0> operand. (See section 0 for definition of *VertStride*) It is ignored if <src0> is an immediate operand.
20	Reserved: MBZ
19:16	**Src0.ChanSel[7:4]** This is bits [7:4] of the *ChanSel* field for <src0> operand. (See section 0 for definition of *ChanSel*).It is ignored if <src0> is an immediate operand.
15	**Src0.AddrMode – Source-0 Address Mode.** This field is the *AddrMode* for <src0> operand. (See section 0 for definition of *AddrMode*) It is ignored if <src0> is an immediate operand.
14:13	**Src0.SrcMod – Source-0 Source Modifier.** This field is the *SrcMod* for source operand <src0>. (See section 0 for definition of *SrcMod*)It is ignored if <src0> is an immediate operand.
12:5	**Src0.RegNum – Source-0 Register Number** This is the *RegNum* field for source operand <src0>. (See section 0 for definition of *RegNum.*) It is ignored if <src0> is an immediate operand.
4	**Src0.SubRegNum[4]** This is the 16-byte aligned sub-register address for source operand <src0>. (See section 0 for definition of *SubRegNum*) It is ignored if <src0> is an immediate operand.
3:0	**Src0.ChanEn – Source-0 Channel Enable** This is the *ChanEn* field for source operand <src0>. (See section 0 for definitions of *ChanEn*) It is ignored if <src0> is an immediate operand.

Table 7-12. Instruction Source-0 Doubleword in Direct+Align1 mode

Bits	Description
31:26	Reserved: MBZ
25	**FlagSubRegNum – Flag Sub-Register Number.** This field specifies the sub-register number for a flag register operand.
24:21	**Src0.VertStride – Source-0 Vertical Stride** This is the *VertStride* field for <src0> operand. (See section 0 for definition of *VertStride*) It is ignored if <src0> is an immediate operand.

Bits	Description
20:18	**Src0.Width.** This is the *Width* field for source operand <src0>. (See section 0 for definition of *Width*) It is ignored if <src0> is an immediate operand.
17:16	**Src0.HorzStride.** This is the *HorzStride* field for source operand <src0>. (See section 0 for definition of *HorzStride*) It is ignored if <src0> is an immediate operand.
15	**Src0.AddrMode – Source-0 Address Mode.** This is the *AddrMode* for source operand <src0>. (See section 0 for definition of *AddrMode*) It is ignored if <src0> is an immediate operand.
14:13	**Src0.SrcMod – Source-0 Source Modifier.** This is the *SrcMod* field for source operand <src0>. (See section 0 for definition of *SrcMod*) It is ignored if <src0> is an immediate operand.
12:5	**Src0.RegNum – Source-0 Register Number.** This is the *RegNum* field for source operand <src0>. (See section 0 for definition of *RegNum.*) It is ignored if <src0> is an immediate operand.
4:0	**Src0.SubRegNum – Source-0 Sub-Register Number.** This is the *SubRegNum* field for source operand <src0>. (See section 0 for definition of *SubRegNum*) It is ignored if <src0> is an immediate operand.

Table 7-13. Instruction Source-0 Doubleword in Indirect+Align16 mode

Bits	Description
31:26	Reserved: MBZ
25	**FlagSubRegNum – Flag Sub-Register Number.** This field specifies the sub-register number for a flag register operand.
24:21	**Src0.VertStride – Source-0 Vertical Stride.** This is the *VertStride* field for <src0> operand. (See section 0 for definition of *VertStride*) It is ignored if <src0> is an immediate operand.
20	Reserved: MBZ
19:16	**Src0.ChanSel[7:4] – Source-0 Channel Select.** This is bits [7:4] of the *ChanSel* field for <src0> operand. (See section 0 for definition of *ChanSel*). It is ignored if <src0> is an immediate operand.
15	**Src0.AddrMode – Source-0 Address Mode.** This is the *AddrMode* for source operand <src0>. (See section 0 for definition of *AddrMode*) It is ignored if <src0> is an immediate operand.
14:13	**Src0.SrcMod – Source-0 Source Modifier.** This is the *SrcMod* field for source operand <src0>. (See section 0 for definition of *SrcMod*) It is ignored if <src0> is an immediate operand.
12:10	**Src0.AddrSubRegNum – Source-0 Address Sub-Register Number.** This is the *AddrSubRegNum* field for source operand <src0>. (See section 0 for definition of *AddrSubRegNum.*) It is ignored if <src0> is an immediate operand.

Bits	Description
9:4	**Src0.AddrImm[9:4] – Source-0 Address Immediate.** This contains the half-register aligned *AddrImm* field ((bits [9:4]) for <src0>. (See section 0 for definition of *AddrImm*) It is ignored if <src0> is an immediate operand.
3:0	**Src0.ChanEn – Source-0 Channel Enable** . This is the *ChanEn* field for source operand <src0>. (See section 0 for definitions of *ChanEn*) It is ignored if <src0> is an immediate operand.

Table 7-14. Instruction Source-0 Doubleword in Indirect+Align1 mode

Bits	Description
31:26	Reserved: MBZ
25	**FlagSubRegNum – Flag Sub-Register Number.** This field specifies the sub-register number for a flag register operand.
24:21	**Src0.VertStride – Source-0 Vertical Stride.** This is the *VertStride* field for <src0> operand. (See section 0 for definition of *VertStride*) It is ignored if <src0> is an immediate operand.
20:18	**Src0.Width.** This is the *Width* field for source operand <src0>. (See section 0 for definition of *Width*) It is ignored if <src0> is an immediate operand.
17:16	**Src0.HorzStride.** This is the *HorzStride* field for source operand <src0>. (See section 0 for definition of *HorzStride*) It is ignored if <src0> is an immediate operand.
15	**Src0.AddrMode – Source-0 Address Mode.** This is the *AddrMode* for source operand <src0>. (See section 0 for definition of *AddrMode*) It is ignored if <src0> is an immediate operand.
14:13	**Src0.SrcMod – Source-0 Source Modifier.** This is the *SrcMod* field for source operand <src0>. (See section 0 for definition of *SrcMod*) It is ignored if <src0> is an immediate operand.
12:10	**Src0.AddrSubRegNum – Source-0 Address Sub-Register Number.** This is the *AddrSubRegNum* field for source operand <src0>. (See section 0 for definition of *AddrSubRegNum.*) It is ignored if <src0> is an immediate operand.
9:0	**Src0.AddrImm – Source-0 Address Immediate.** This is the byte aligned *AddrImm* field for <src0>. (See section 0 for definition of *AddrImm*) It is ignored if <src0> is an immediate operand.

7.2.4.2 3-src Instructions

This section describes the field in DW2 and DW3 of the 3-src instruction format.

Table 7-15. Instruction DW2 and DW3

DW	Bits	Description
DW3	31:30	Reserved: MBZ
	29:22	**Source2 Register Number.** This field contains the register number for source2.
	21:19	**Source2 Subregister Number.** This field contains the subregister number for source2.
	18:11	**Source2 Swizzle.** This field contains the swizzle control for source2. Refer to Table.5-5 for encoding.
	10:10	**Source2 Replication Control.** This field controls replication for source2. Refer to Table.5-5 for encoding.
	9:9	Reserved: MBZ
	8:1	**Source1 Register Number.** This field contains the register number for source1.

DW	Bits	Description
	0	**Source1 Subregister Number.** This field contains the subregister number for source1.
DW2	31:30	**Source1 Subregister Number.** This field contains the subregister number for source1.
	29:22	**Source1 Swizzle.** This field contains the swizzle control for source1. Refer to Table.5-5 for encoding.
	21:21	**Source1 Replication Control.** This field controls replication for source1. Refer to Table.5-5 for encoding.
	20:20	Reserved: MBZ
	19:12	**Source0 Register Number.** This field contains the register number for source0.
	11:9	**Source0 Subregister Number.** This field contains the subregister number for source0.
	8:1	**Source0 Swizzle.** This field contains the swizzle control for source0. Refer to Table.5-5 for encoding.
	0:0	**Source0 Replication Control.** This field controls replication for source0. Refer to Table.5-5 for encoding.

7.2.5 Instruction Source-1 Doubleword (DW3)

Source-1 Doubleword (DW3) contains the second source operand (<src1>) and is used to hold the 32-bit immediate source (imm32 as <src0> or <src1>). Table 7-16 and Table 7-17 define the fields in this doubleword with the following exceptions:

- If <src0> is an immediate operand, this doubleword contains **imm32** for <src0>.
- If <src1> is an immediate operand, this doubleword contains **imm32** for <src1>.
- If the instruction is a send, bit 31 of this doubleword contains **EOT** field.
    - If <src1> is immediate, the remaining 31 bits in this doubleword is **MsgDescpt31**.
    - If <src1> is a register, <src1> must be a0.0. The rest of this doubleword will be configured accordingly.
- If indirect address is supported for <src1>, Table 7-18 and Table 7-19 define the fields in DW3 for indirectly addressed <src1> in Align16 and Align1 modes.

Table 7-16. Instruction Source-1 Doubleword in Direct + Align16 mode

Bits	Description
31:25	Reserved: MBZ
24:21	**Src1.VertStride – Source-1 Vertical Stride.** This field is the *VertStride* for <src1> operand. (See section 0 for definition of *VertStride*) It is ignored if <src1> is an immediate operand.
20	Reserved: MBZ

Bits	Description
19:16	**Src1.ChanSel[7:4]** This contains bits [7:6] of the *ChanSel* field for <src1> operand. (See section 0 for definition of *ChanSel*) It is ignored if <src1> is an immediate operand.
15	Reserved: MBZ
14:13	**Src1.SrcMod – Source-1 Source Modifier.** This field is the *SrcMod* for <src1> operand. (See section 0 for definition of *SrcMod*) It is ignored if <src1> is an immediate operand.
12:5	**Src1.RegNum.** This field is the *RegNum* field for <src1> operand. (See section 0 for definition of *RegNum.*) It is ignored if <src1> is an immediate operand.
4	**Src1.SubRegNum[4].** This field is bit [4] of the *SubRegNum* field for <src1>. (See section 0 for definition of *SubRegNum*) It is ignored if <src1> is an immediate operand.
3:0	**Src1.ChanEn – Source-1 Channel Enable.** It is the channel enable field for <src1>. (See section 0 for definitions of *ChanEn*)It is ignored if <src1> is an immediate operand.

Table 7-17. Instruction Source-1 Doubleword in Direct + Align1 mode

Bits	Description
31:25	Reserved: MBZ
24:21	**Src1.VertStride – Source-1 Vertical Stride.** This field is the *VertStride* for <src1> operand. (See section 0 for definition of *VertStride*) It is ignored if <src1> is an immediate operand.
20:18	**Src1.Width.** This is the *Width* field for source operand <src1>. (See section 0 for definition of *Width*) It is ignored if <src1> is an immediate operand.
17:16	**Src1.HorzStride.** This is the *HorzStride* field for source operand <src1>. (See section 0 for definition of *HorzStride*) It is ignored if <src1> is an immediate operand.
15	Reserved: MBZ
14:13	**Src1.SrcMod – Source-1 Source Modifier.** This field is the *SrcMod* for <src1> operand. (See section 0 for definition of *SrcMod*) It is ignored if <src1> is an immediate operand.
12:5	**Src1.RegNum – Source-1 Register Number.** This is the *RegNum* field for source operand <src1>. (See section 0 for definition of *RegNum.*) It is ignored if <src1> is an immediate operand.
4:0	**Src1.SubRegNum – Source-1 Sub-Register Number.** This is the *SubRegNum* field for source operand <src1>. (See section 0 for definition of *SubRegNum*) It is ignored if <src1> is an immediate operand.

Table 7-18. Instruction Source-1 Doubleword in Indirect+Align16 mode

Bits	Description
31:25	Reserved: MBZ
24:21	**Src1.VertStride – Source-1 Vertical Stride** This is the *VertStride* field for <src1> operand. (See section 0 for definition of *VertStride*) It is ignored if <src1> is an immediate operand.
20	Reserved: MBZ
19:16	**Src1.ChanSel[7:4] – Source-1 Channel Select** This is bits [7:4] of the *ChanSel* field for <src1> operand. (See section 0 for definition of *ChanSel*). It is ignored if <src1> is an immediate operand.
15	**Src1.AddrMode – Source-1 Address Mode** This is the *AddrMode* for source operand <src1>. (See section 0 for definition of *AddrMode*) It is ignored if <src1> is an immediate operand.
14:13	**Src1.SrcMod – Source-1 Source Modifier** This is the *SrcMod* field for source operand <src1>. (See section 0 for definition of *SrcMod*) It is ignored if <src1> is an immediate operand.
12:10	**Src1.AddrSubRegNum – Source-1 Address Sub-Register Number** This is the *AddrSubRegNum* field for source operand <src1>. (See section 0 for definition of *AddrSubRegNum.*) It is ignored if <src1> is an immediate operand.
9:4	**Src1.AddrImm[9:4] – Source-1 Address Immediate** This contains the half-register aligned *AddrImm* field ((bits [9:4]) for <src1>. (See section 0 for definition of *AddrImm*) It is ignored if <src1> is an immediate operand.
3:0	**Src1.ChanEn – Source-1 Channel Enable** This is the *ChanEn* field for source operand <src1>. (See section 0 for definitions of *ChanEn*) It is ignored if <src1> is an immediate operand.

Table 7-19. Instruction Source-1 Doubleword in Indirect+Align1 mode

Bits	Description
31:25	Reserved: MBZ
24:21	**Src1.VertStride – Source-1 Vertical Stride** This is the *VertStride* field for <src1> operand. (See section 0 for definition of *VertStride*) It is ignored if <src1> is an immediate operand.
20:18	**Src1.Width** This is the *Width* field for source operand <src1>. (See section 0 for definition of *Width*) It is ignored if <src1> is an immediate operand.
17:16	**Src1.HorzStride** This is the *HorzStride* field for source operand <src1>. (See section 0 for definition of *HorzStride*) It is ignored if <src1> is an immediate operand.

Bits	Description
15	**Src1.AddrMode – Source-1 Address Mode** This is the *AddrMode* for source operand <src1>. (See section 0 for definition of *AddrMode*) It is ignored if <src1> is an immediate operand.
14:13	**Src1.SrcMod – Source-1 Source Modifier** This is the *SrcMod* field for source operand <src1>. (See section 0 for definition of *SrcMod*) It is ignored if <src1> is an immediate operand.
12:10	**Src1.AddrSubRegNum – Source-1 Address Sub-Register Number** This is the *AddrSubRegNum* field for source operand <src1>. (See section 0 for definition of *AddrSubRegNum.*) It is ignored if <src1> is an immediate operand.
9:0	**Src1.AddrImm – Source-1 Address Immediate** This is the byte aligned *AddrImm* field for <src1>. (See section 0 for definition of *AddrImm*) It is ignored if <src1> is an immediate operand.

7.3 Opcode Encoding

Byte 0 of the 128-bit instruction word contains the opcode. The opcode uses 7 bits. Bit location 7 in byte 0 is reserved for future opcode extension.

There are total of 48 opcodes defined. These opcodes are encoded and organized into five groups based on the type of operations: Special instructions, move/logic instructions (opcode=00xxxxxb), flow control instructions (opcode=010xxxxb), miscellaneous instructions (opcode=011xxxxb), parallel arithmetic instructions (opcode=100xxxxb), and vector arithmetic instructions (opcode=101xxxxb). Opcodes 110xxxb are reserved.

7.3.1 Move and Logic Instructions

This instruction group has an opcode format of 00xxxxxb.

- The opcodes for move instructions (***mov***, ***sel*** and ***movi***) share the common 5 MSBs in the form of 00000xxb.
- The opcodes for logic instructions (***not***, ***and***, ***or***, and ***xor***) share the common 5 MSBs in the form of 00001xxb.
- The opcodes for shift instructions (***shr***, ***shl***, and ***asr***) share the common 4 MSBs in the form of 0001xxxb. Bit 2 indicates arithmetic or logic shift (0 = logic, 1 = arithmic). Bit 1 is always 0 (which is reserved for future extension to support rotation shift as 0 = shift, 1 = rotate). Bit 0 indicates the shift direction (0 = right, 1 = left).
- The opcodes for compare instructions (***cmp*** and ***cmpn***) share the common 6 MSBs in the form of 001000xb. Bit 0 indicates whether it is a normal compare, *cmp*, or a special compare-NaN, *cmpn*.
- This group of instructions does not implicitly update the accumulators.
- Instruction compression applies to this group.

Table 7-20. Move and Logic Instructions

Opcode		Instruction	Description	#src	#dst
dec	hex				
1	0x01	***mov***	Component-wise move	1	1
2	0x02	***sel***	Component-wise selective move based on predication	2	1
3	0x03	***movi***	Fast component-wise indexed move	1	1
4	0x04	***not***	Component-wise one's compliment (bitwise not)	1	1
5	0x05	***and***	Component-wise logical AND (bitwise and)	2	1
6	0x06	***or***	Component-wise logical OR (bitwise or)	2	1
7	0x07	***xor***	Component-wise logical XOR (bitwise xor)	2	1
8	0x08	***shr***	Component-wise logical shift right	2	1
9	0x09	***shl***	Component-wise logical shift left	2	1
10	**0x0A**	***dim***	**Double Precision Floaitng Point Immediate Data Move**	**1**	**1**
11	0x0A-0x0B	*Reserved*			
12	0x0C	***asr***	Component-wise arithmetic shift right	2	1
13-**15**	0x0D-0x0F	*Reserved*			
16	0x10	***cmp***	Component-wise compare, store condition code in destination	2	1
17	0x11	***cmpn***	Component-wise compare-NaN, store condition code in destination	2	1
18	0x12	*Reserved*			
19	0x13	***Reserved***			
20	0x14	***Reserved***			
21-22	0x15-0x16	*Reserved*			
23	0x17	***Reserved***			
24	0x18	***Reserved***			
25	0x19	***Reserved***			
26	0x1A	***Reserved***			
20-31	0x12-0x1F	*Reserved*			

7.3.2 Flow Control Instructions

This instruction group has an opcode format of 010xxxxb.

- This group of instructions does not implicitly update the accumulators.
- Instruction compression is not allowed for this group.

Table 7-21. Flow Control Instructions

Opcode		Instruction	Description	#src	#dst
dec	hex				
32	0x20	***jmpi***	Jump indexed	1	0
33	0X21	***Reserved***			
34	0x22	***if***	If	0/2	0
35	0X23	***Reserved***			
36	0x24	***else***	Else	1	0
37	0x25	***endif***	End if	0	0
38	0x26	***case***	Case – Inside Switch block	0/2	0
39	0x27	***while***	While	1	0
40	0x28	***break***	Break	1	0
41	0x29	***cont***	Continue	1	0
42	0x2A	***halt***	Halt	1	0
43	0x2B	*Reserved*			
44	0x2C	***call***	Subroutine call	1	1
45	0x2D	***return***	Subroutine return	1	1
46	0x2E	***fork***	go into 16pixel execution mode from 32pixel execution mode	1	0
47	0x2F	*Reserved*			

7.3.3 Miscellaneous Instructions

This instruction group has an opcode format of 011xxxxb.

- This group of instructions does not implicitly update the accumulators.
- Instruction compression is not allowed for this group.

Table 7-22. Miscellaneous Instructions

Opcode		Instruction	Description	#src	#dst
dec	hex				
48	0x30	***wait***	Wait for (external) notification	1	0
49	0x31	***send***	Send	1	1
50	0x32	***sendc***	Conditional Send (based on TDR)	1	1
51-55	0x33-0x37	*Reserved*			
56	0x38	***math***	Math functions for extended math pipeline	1/2	1/2
57-63	0x39-0x3F	*Reserved*			

7.3.4 Parallel Arithmetic Instructions

This instruction group has an opcode format of 100xxxxb.

- The opcode for round instructions (***rndu***, ***rndd***, ***rnde***, and ***rndz***) share the common 5 MSBs in the form of 10001xxb, with the lower 2 bits indicate the type of round.
- These instructions implicitly update the accumulators if the Accumulator Disable bit in control register cr0.0 not set.
    - Some instructions such as ***frc***, ***lzd***, etc, perform the operation after the accumulator. Therefore, when the accumulator is implicitly updated, the content is undefined. Details can be found in ISA Reference Chapter.
- Instruction compression applies to this group.

Table 7-23. Parallel Arithmetic Instructions

Opcode		Instruction	Description	#src	#dst
dec	hex				
64	0x40	***add***	Component-wise addition	2	1
65	0x41	***mul***	Component-wise multiply	2	1
66	0x42	***avg***	Component-wise average of the two source operands	2	1
67	0x43	***frc***	Component-wise floating point truncate-to-minus-infinity fraction	1	1
68	0x44	***rndu***	Component-wise floating point rounding up (ceiling)	1	1
69	0x45	***rndd***	Component-wise floating point rounding down (floor)	1	1
70	0x46	***rnde***	Component-wise floating point rounding toward nearest even	1	1

Opcode		Instruction	Description	#src	#dst
dec	hex				
71	0x47	***rndz***	Component-wise floating point rounding toward zero	1	1
72	0x48	***mac***	Component-wise multiply accumulate	2	1
73	0x49	***mach***	multiply accumulate high	2	1
74	0x4A	***lzd***	leading zero detection	1	1
75	0x4B	***Reserved***			
76	0x4C	***Reserved***			
77	0x4D	***Reserved***			
78	0x4E	***Reserved***			
79	0x4F	***Reserved***			

7.3.5 Vector Arithmetic Instructions

- This instruction group has an opcode format of 101xxxxb.
- These instructions implicitly update the accumulators if the Accumulator Disable bit in control register cr0.0 not set.
    - Some instructions such as ***dp4-dp2***, etc, perform the operation after the accumulator. Therefore, when the accumulator is implicitly updated, the content is undefined. Details can be found in ISA Reference Chapter.
- Instruction compression applies to this group.

Table 7-24. Vector Arithmetic Instructions

Opcode		Instruction	Description	#src	#dst
dec	hex				
80	0x50	***sad2***	2-wide sum of absolute difference	2	1
81	0x51	***sada2***	2-wide sad accumulate	2	1
82-83	0x52-0x53	*reserved*			
84	0x54	***dp4***	4-wide dot product for 4-vector	2	1
85	0x55	***dph***	4-wide homogenous dot product for 4-vector	2	1
86	0x56	***dp3***	3-wide dot product for 4-vector	2	1
87	0x57	***dp2***	2-wide dot product for 4-vector	2	1
88	0x58	*reserved*			

Opcode		Instruction	Description	#src	#dst
dec	hex				
89	0x59	***line***	Component-wise line equation computation (a multiply-add)	2	1
90	0x5A	***pln***	Component-wise floating point plane equation computation (a multiply-multiply-add)	2(3)	1
91	0x5B	***mad***	Component-wise floating point mad computation (a multiple-add)	3	1
92	0x5C	***lrp***	Component-wise floating point lrp computation (blend)	3	1
93-95	0x5D-0x5F	*reserved*			

7.3.6 Special Instructions

There are two special instructions, namely, *nop* (opcode = 0x7E) and *illegal* (opcode = 0x**00**).

- *Nop* instruction may be used for instruction padding in memory between two normal instructions to force alignment or to introduce instruction execution delay. Currently, there is no need for between-instruction padding.
- *Illegal* instruction may be used for instruction padding in memory outside the normal instruction sequence such as before or after the kernel program as well as between subroutines.
- *Nop* and *illegal* instructions do not have source operands or destination operand. Therefore, they do not implicitly update the accumulator register. They cannot be compressed.

Table 7-25. Special Instructions

Opcode		Instruction	Description	#src	#dst
dec	hex				
0	0x00	***illegal***	Illegal instruction	0	0
96-124	0x60-0x7C	*Reserved*			
125	0x7D	***nenop***	Non-executed No-op	0	0
126	0x7E	***nop***	No-op	0	0
127	0x7F	*Reserved*	(may be used as an extension code)		

7.4 Native Instruction BNF

The Backus-Naur Form (BNF) grammar identifies the assembly language syntax, which is native to the hardware. It does not include intelligent defaults, assembler pragmas, etc.

7.4.1 Instruction Groups

```
<Instruction>          ::=  <UnaryInstruction>
                          | <BinaryAccInstruction>
                          | <BinaryInstruction>
                          | <TriInstruction>
                          | <JumpInstruction>
                          | <BranchLoopInstruction>
                          | <ElseInstruction>
                          | <BreakInstruction>
                          | <MaskControlInstruction>
                          | <SyncInstruction>
                          | <SpecialInstruction>

<UnaryInstruction>     ::=  <Predicate> <UnaryInst> <ExecSize> <Dst> <SrcAccImm> <InstOptions>

<UnaryInst>            ::=  <UnaryOp> <ConditionalModifier> <Saturate>

<UnaryOp>              ::=  "mov" | "frc" | "rndu" | "rndd" | "rnde" | "rndz" | "not" | "lzd"

<BinaryInstruction>    ::=  <Predicate> <BinaryInst> <ExecSize> <Dst> <Src> <SrcImm> <InstOptions>

<BinaryInst>           ::=  <BinaryOp> <ConditionalModifier> <Saturate>

<BinaryOp>             ::=  "mul" | "mac" | "mach" | "line"

                          | "sad2" | "sada2" | "dp4" | "dph" | "dp3" | "dp2"

<BinaryAccInstruction> ::=  <Predicate> <BinaryAccInst> <ExecSize> <Dst> <SrcAcc> <SrcImm>
                            <InstrOptions>

<BinaryAccInst>        ::=  <BinaryAccOp> <ConditionalModifier> <Saturate>

<BinaryAccOp>          ::=  "avg" | "add" | "sel"

                          | "and" | "or" | "xor"

                          | "shr" | "shl" | "asr"

                          | "cmp" | "cmpn"

<TriInstruction>       ::=  <Predicate> <TriInst> <ExecSize> <PostDst> <CurrDst> <TriSrc>
                            <MsgDesc> <InstOptions>
```

<TriInst>	::=	<TriOp> <ConditionalModifier> <Saturate>
<TriOp>	::=	"**send**"
<JumpInstruction>	::=	<JumpOp> <RelativeLocation2>
<JumpOp>	::=	"**jmpi**"
<BranchLoopInstruction>	::=	<Predicate> <BranchLoopOp> < RelativeLocation>
<BranchLoopOp>	::=	"**if**" | "**iff**" | "**while**"
<ElseInstruction>	::=	<ElseOp> < RelativeLocation>
<ElseOp>	::=	"**else**"
<BreakInstruction>	::=	<Predicate> <BreakOp> <LocationStackCtrl>
<BreakOp>	::=	"**break**" | "**cont**" | "**halt**"
<SyncInstruction>	::=	<Predicate> <SyncOp> <NotifyReg>
<SyncOp>	::=	"**wait**"
<SpecialInstruction>	::=	"**do**" | "**endif**" |"**nop**" | "**illegal**"

7.4.2 Destination Register

<Dst>	::=	<DstOperand>
	|	<DstOperandEx>
<DstOperand>	::=	<DstReg> <DstRegion> <WriteMask> <DstType>
<DstOperandEx>	::=	<AccReg> <DstRegion> <DstType>
	|	<FlagReg> <DstRegion> <DstType>
	|	<AddrReg> <DstRegion> <DstType>
	|	<MaskReg> <DstRegion> <DstType>
	|	<MaskStackReg>

| <ControlReg>

| <IPReg>

| <NullReg>

<DstReg> ::= <DirectGenReg> | <IndirectGenReg>

| <DirectMsgReg> | <IndirectMsgReg>

<PostDst> ::= <PostDstReg> <DstRegion> <WriteMask> <DstType>

| <NullReg>

<PostDstReg> ::= <DirectGenReg> | <IndirectGenReg>

<CurrDst> ::= <DirectAlignedMsgReg>

7.4.3 Source Register

Source with Accumulator Access and with Immediate

<SrcAccImm> ::= <SrcAcc>
| <Imm32> <SrcImmType>

<SrcAcc> ::= <DirectSrcAccOperand>
| <IndirectSrcOperand>

<DirectSrcAccOperand> ::= <DirectSrcOperand>

| <SrcArcOperandEx>

| <AccReg> <SrcType>

<SrcArcOperandEx> ::= <FlagReg> <Region> <SrcType>

| <AddrReg> <Region> <SrcType>

| <MaskReg> <Region> <SrcType> [Pre-DevSNB]

| <MaskStackReg> [Pre-DevSNB]

| <ControlReg>

| <StateReg>

| <NotifyReg>

| <IPReg>

| <NullReg>

<IndirectSrcOperand> ::= <SrcModifier> <IndirectGenReg> <IndirectRegion> <Swizzle > <SrcType>

Source without Accumulator Access

<Src> ::= <DirectSrcOperand>
| <IndirectSrcOperand>

< DirectSrcOperand > ::= <SrcModifier> <DirectGenReg> <Region> <Swizzle> <SrcType>

| <SrcArcOperandEx>

<TriSrc> ::= <SrcModifier> <DirectGenReg> <Region> <Swizzle> <SrcType>
| <NullReg>

<MsgDesc> ::= <ImmDesc>
| <Reg32>

<Reg32> ::= <DirectGenReg> <Region> <SrcType>

Source without Accumulator Access or IP Access

<SrcImm> ::= <DirectSrcOperand>
| <Imm32> <SrcImmType>

7.4.4 Address Registers

<AddrParam> ::= <AddrReg> <ImmAddrOffset>

<ImmAddrOffset> ::= “”

| “,” <ImmAddrNum>

7.4.5 Register Files and Register Numbers

<DirectGenReg>	::=	<GenRegFile> <GenRegNum> <GenSubRegNum>
<IndirectGenReg>	::=	<GenRegFile> "**[**" <AddrParam> "**]**"
<GenRegFile>	::=	"**r**"
<GenRegNum>	:: =	"**0**"…"**127**"
<GenSubRegNum>	:: =	""
	|	"**.0**"..."**.7**"
	|	"**.0**"..."**.15**"
	|	"**.0**"..."**.31**"
<DirectMsgReg>	::=	<DirectAlignedMsgReg> <MsgSubRegNum>
<DirectAlignedMsgReg>	::=	<MsgRegFile> <MsgRegNum>
<IndirectMsgReg>	::=	<MsgRegFile> "**[**" <AddrParam> "**]**"
<MsgRegFile>	::=	"**m**"
<MsgRegNum>	:: =	"**0**"…"**15**"
<MsgSubRegNum>	:: =	<GenSubRegNum>
<AddrReg>	::=	<AddrRegFile> <AddrSubRegNum>
<AddrRegFile>	::=	"**a0**"
<AddrSubRegNum>	:: =	""
	|	"**.0**" … "**.7**"
<AccReg>	::=	"**acc**" <AccRegNum><AccSubRegNum>
<AccRegNum>	:: =	"**0**" | "**1**"
<AccSubRegNum>	:: =	<GenSubRegNum>
<FlagReg>	::=	"**f0**" <FlagSubRegNum>
<FlagSubRegNum>	:: =	""
	|	".**0**"..."**.1**"

<NotifyReg>	::=	"**n**" <NotifyRegNum>
<NotifyRegNum>	:: =	"**0**"..."**2**"
<StateReg>	::=	"**sr0**" <StateSubRegNum>
<StateSubRegNum>	:: =	".**0**"... ".**1**"
<ControlReg>	::=	"**cr0**" <ControlSubRegNum>
<ControlSubRegNum>	:: =	".**0**" ..."**.2**"
<IPReg>	::=	"**ip**"
<NullReg>	::=	"**null**"

7.4.6 Relative Location and Stack Control

<RelativeLocation>	::=	<imm16>
<RelativeLocation2>	::=	<imm32> | <reg32>
<LocationStackCtrl>	::=	<imm32>

7.4.7 Regions

<DstRegion>	::=	"**<**" <HorzStride> "**>**"
<IndirectRegion>	::=	<Region> | <RegionWH> | <RegionV>
<Region>	::=	"**<**" <VertStride> "**;**" <Width> "**,**" <HorzStride> "**>**"
<RegionWH>	::=	"**<**" <Width> "**,**" <HorzStride> "**>**"
<RegionV>	::=	"**<**"<VertStride> "**>**"
<VertStride>	::=	"**0**" | "**1**" | "**2**" | "**4**" | "**8**" | "**16**" | "**32**"
<Width>	::=	"**1**" | "**2**" | "**4**" | "**8**" | "**16**"

<HorzStride> ::= “**0**” | “**1**” | “**2**” | “**4**”

7.4.8 Types

<SrcType> ::= “**:f**” | “**:ud**” | “**:d**” | “**:uw**” | “**:w**” | “**:ub**” | “**:b**”

<SrcImmType> ::= <SrcType> | “**:v**” | “**:vf**”

<DstType> ::= <SrcType>

7.4.9 Write Mask

<WriteMask> ::= “”

| “.” “**x**” | “.” “**y**” | “.” “**z**” | “.” “**w**”

| “.” “**xy**” | “.” “**xz**” | “.” “**xw**” | “.” “**yz**” | “.” “**yw**” | “.” “**zw**”

| “.” “**xyz**” | “.” “**xyw**” | “.” “**xzw**” | “.” “**yzw**”

| “.” “**xyzw**”

7.4.10 Swizzle Control

<Swizzle> ::= “”

| “.” <ChanSel>

| “.” <ChanSel> <ChanSel> <ChanSel> <ChanSel>

<ChanSel> ::= “**x**” | “**y**” | “**z**” | “**w**”

7.4.11 Immediate Values

<ImmAddrNum> ::= “**-512**”… “**511**”

<Imm32> ::= “**0.0**”… “**±1.0*2**$^{-128...127}$” | “**0**”…”**2**^{32}**-1**” | “**-2**^{31}”…”**2**^{31}**-1**”

<Imm16> ::= “**0**”…”**2**^{16}**-1**” | “**-2**^{15}”…”**2**^{15}**-1**”

<ImmDesc> ::= “**0**”…”**2**^{32}**-1**”

7.4.12 Predication and Modifiers

Instruction Predication

<Predicate> ::= “”

| “**(**” <PredState> <FlagReg> <PredCntrl> “**)**”

<PredState> ::= “”

| “**+**”

| “**-**“

<PredCntrl> ::= “”

| “**.x**” | “**.y**” | “**.z**” | “**.w**”

| “**.any2h**” | “**.all2h**”

| “**.any4h**” | “**.all4h**”

| “**.any8h**” | “**.all8h**”

| “**.any16h**” | “**.all16h**”

| “**.anyv**” | “**.allv**”

Source Modification

<SrcModifier**>** ::= “”

| “**-**”

| “**(abs)**”

| “**-**” “**(abs)**”

Instruction Modification

<ConditionalModifier> ::= “”

| <CondMod> “**.** ” <FlagReg>

<CondMod> ::= “**.z**” | “**.e**” | “**.nz**” | “**.ne**” | “**.g**” | “**.ge**” | “**.l**” | “**.le**” | “**.o**” | “**.r**” | “**.u**”

<Saturate> ::= “”

| “**.sat**”

Execution Size

<ExecSize>	::=	"**(**" <NumChannels> "**)**"
<NumChannels>	::=	"**1**" | "**2**" | "**4**" | "**8**" | "**16**" | "**32**"

7.4.13 Instruction Options

<InstOptions>	::=	""
	|	"**{**" <InstOption> "**}**"
	|	"**{**" <InstOption> <InstOptionEx> "**}**"
<InstOptionEx>	::=	""
	|	"**,**" <InstOption> <InstOptionEx>
<InstOption>	::=	<AccessMode>
	|	<ComprCtrl>
	|	<ThreadCtrl>
	|	<DependencyCtrl>
	|	<MaskCtrl> [Pre-DevSNB]
	|	<SendCtrl>
	|	<AccWrCtrl> [DevSNB+]
	|	<WECtrl> [DevSNB+]
<AccessMode>	::=	"**Align1**" | "**Align16**"
<ComprCtrl>	::=	"**SecHalf**" | "**Compr**"
<ThreadCtrl>	::=	"**Switch**"
<DependencyCtrl>	::=	"**NoDDChk**" | "**NoDDClr**"
<MaskCtrl>	::=	"**NoMask**"
<SendCtrl>	::=	"**EOT**"

Note for Assembler: Compression control "Compr" has a direct map to the binary instruction word. It may be omitted as long as the Assembler is able to determine whether an instruction is in compressed mode or not based on the execution size and the mode of operation.

7.5 Deprecated Features

7.5.1 Defeatured Instructions

The following instructions are removed from implementation mainly due to implementation cost/schedule reasons. They are candidates for future generations.

- Sum of Absolute Difference 4 (sad4)
- Sum of Absolute Difference Accumulate 4 (sada4)
- Add Accumulate (aac)
- Min (min)
- Max (max)
- Next (next)
- Swizzle (swz)
- Dot Product Accumulate 2 (dpa2)
- Rotation Shift Left (rsl)
- Rotation Shift Right (rsr)

7.5.2 Others

The following features are also deprecated from implementation.

- Restricted 16-bit Half Floating-Point Numbers

8. Instruction Set Reference

This chapter describes the functions of instructions. Each instruction is given a different page and the pages are sorted in alphabetical order according to assembly language mnemonic.

8.1 Conventions

8.1.1 Pseudo Code Format

The instructions are explained in the following pseudo-code format that resembles the assembly instruction format.

```
[(<pred>)] opcode (<exec_size>) <dst> <src0> [<src1>]
```

Square bracket "[ ]" is used to signify that the field is optional. Saturation modifier and instruction options are omitted for simplicity.

8.1.2 General Macros and Definitions

INST_BYTE_COUNT is defined as a constant of 16 bytes.

```
#define INST_BYTE_COUNT          16       // byte count of instruction word
```

Function *floor()* converts a floating point value to an integral floating point value. For a given floating point value, from its closest two integral float values, function *floor()* returns the one that is closer to the negative infinity. For example, *floor*(1.3f) = 1.0f, and *floor*(-1.3f) = -2.0f.

```
float floor(float g)
{
        return maximum( any integral float f: f <= g)
}
```

Function *Condition()* takes the conditional signals {SN, ZR, OF, IN, NC} of *result*, generates a Boolean data according to a conditional evaluation controlled by the conditional modifier *cmod*, and returns the Boolean data.

```
Bool Condition(result, cmod)
{
}
```

Function *ConditionNaN()* takes the conditional signals {SN, ZR, OF, IN, NC, NS} of *result*, generates a Boolean data according to a conditional evaluation controlled by the conditional modifier *cmod*, and

returns the Boolean data. The only difference between *Condition()* and *ConditionNaN()* is that *ConditionNaN()* uses the NS (NaN of the second source) signal.

```
Bool ConditionNaN(result, cmod)
{
}
```

Function *Jump()* jumps the instruction sequence from the current instruction location by *InstCount* number of instructions. If *InstCount* is a positive number, it jumps forward; if *InstCount* is a negative number, it jumps backward; if *InstCount* is zero, it is effectively an infinite loop on the current instruction.

```
void Jump(int InstCount)
{
        IP = IP + (InstCount * INST_BYTE_COUNT)
}
```

8.2 Evaluate Write Enable

The WrEn should be evaluated as below.

```
if (WECtrl == 1) {
        for (n =0; n < exec_size; n++) {
                WrEn[n] = 1;
        }
} else {
        for (n =0; n < exec_size; n++) {
                if (PcIP[n] == ExIP) {
                        WrEn[n] = 1;
                } else {
                        WrEn[n] = 0;
                }
        }
}

if (PredCtrl != “0000”) {
        for (n =0; n < exec_size; n++) {
                WrEn[n] = WrEn[n] & PMask[n];
        }
}
```

```
for (n = exec_size; n < 32; n++) {
        WrEn[n] = 0;
}
```

8.3 Instruction Description

The rest of the chapter contains the description of instructions.

8.3.1 add – Addition

Opcode	Instruction	Description
64 (0x40)	add <dst> <src0> <src1>	Component-wise addition of <src0> and <src1> and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[FLT]	[FLT]
•	•	•	•	[INT]	[INT]
•	•	•	•	[INT]	[FLT]

Format:

```
[(<pred>)] add[.<cmod>] (<exec_size>) <dst> <src0> <src1>
```

Syntax:

```
[(<pred>)] add[.<cmod>] (<exec_size>) reg reg reg
[(<pred>)] add[.<cmod>] (<exec_size>) reg reg imm32
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = src0.chan[n] + src1.chan[n];
        }
}
```

Description:

The *add* instruction performs component-wise addition of <src0> and <src1> and stores the results in <dst>.

Addition of two floating point numbers follows rules in Table 8-1 (or

Table 8-2), if the current floating point mode is IEEE mode (or ALT mode).

Table 8-1. Floating point addition of A (column) and B (row) in IEEE mode

	–inf	**–finite**	**–denorm**	**–0**	**+0**	**+denorm**	**+finite**	**+inf**	**NaN**
–inf	–inf	–inf	–inf	–inf	–inf	–inf	–inf	NaN	NaN
–finite	–inf	*	A	A	A	A	**	+inf	NaN
–denorm	–inf	B	–0	–0	+0	+0	B	+inf	NaN
–0	–inf	B	–0	–0	+0	+0	B	+inf	NaN
+0	–inf	B	+0	+0	+0	+0	B	+inf	NaN
+denorm	–inf	B	+0	+0	+0	+0	B	+inf	NaN
+finite	–inf	**	A	A	A	A	***	+inf	NaN
+inf	NaN	+inf	+inf	+inf	+inf	+inf	+inf	+inf	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Notes:

* Result can be { –finite}

** Result can be {–finite, –0, +0, +finite}

*** Result can be { +finite}

Table 8-2. Floating point addition of A (column) and B (row) in ALT mode

	– fmax	**–finite**	**–denorm**	**–0**	**+0**	**+denorm**	**+finite**	**+ fmax**	********
–fmax	–fmax	–fmax	–fmax	–fmax	–fmax	–fmax	–finite	+0	
–finite	–fmax	*	A	A	A	A	**	+fmax	
–denorm	–fmax	B	–0	–0	+0	+0	B	+fmax	
–0	–fmax	B	–0	–0	+0	+0	B	+fmax	
+0	–fmax	B	+0	+0	+0	+0	B	+fmax	
+denorm	–fmax	B	+0	+0	+0	+0	B	+fmax	
+finite	–finite	**	A	A	A	A	***	+fmax	
+fmax	+0	+fmax	+fmax	+fmax	+fmax	+fmax	+fmax	+fmax	
********									

Notes:

* Result can be { –fmax, –finite}

** Result can be {–finite, –0, +0, +finite}

*** Result can be { +fmax, +finite}

**** Result is undefined If any of A and/or is {–inf, +inf, NaN}

8.3.2 and – Logical And

Opcode	Instruction	Description
5 (0x05)	and <dst> <src0> <src1>	Performing component-wise logic AND of <src0> and <src1> and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•		•	•	[INT]	[INT]

Format:

[(<pred>)] and[.<cmod>] (<exec_size>) <dst> <src0> <src1>

Syntax:

[(<pred>)] and[.<cmod>] (<exec_size>) reg reg reg
[(<pred>)] and[.<cmod>] (<exec_size>) reg reg imm32

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = src0.chan[n] & src1.chan[n];
        }
}
```

Description:

The *and* instruction performs component-wise logic AND operation between <src0> and <src1> and stores the results in <dst>. Source modifiers are allowed.

Accumulator register is allowed to be the destination of this instruction with the restrictions listed below.

Restrictions:

Sign (SN) and Overflow (OF) conditions are undefined for this logic instruction. Consequently, saturation modifier (.sat) is not allowed.

This instruction only applies to integer data types. The behavior is undefined if any operand is float.

Accumulator is allowed to be the source of this instruction, but source modifier is not allowed for an accumulator source.

When accumulator is the destination of this instruction, only the low bits corresponding to the data type (16 bits for word or 32 bits for dword integer instruction) in the accumulator contain the correct results. The internal extra-precision bits as well as the sign bit of the accumulator are undefined. Consequently, there are restrictions for subsequent instructions that use the data in the accumulator register created from the previous logical instruction.

- Only logical and data move instructions are allowed to source the accumulator. Results of other instructions (e.g. arithmetic or shift) are undefined.
- When the accumulator is the source of a data move (mov or sel) instruction, the destination operand must be of integer type (e.g. no conversion to float) and this instruction cannot have satuation instruction modifier.

8.3.3 asr – Arithmetic Shift Right

Opcode	Instruction	Description
12 (0x0C)	asr <dst> <src0> <src1>	Performing component-wise arithmetic right shift of <src0> and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[INT]	[INT]

Format:

[(<pred>)] asr[.<cmod>] (<exec_size>) <dst> <src0> <src1>

Syntax:

[(<pred>)] asr[.<cmod>] (<exec_size>) reg reg reg
[(<pred>)] asr[.<cmod>] (<exec_size>) reg reg imm32

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.channel[n] == 1) {
                if (src0.chan[n] >= 0) {
                        dst.chan[n] = src0.chan[n] >> src1.chan[n];
                } else {
                        int maskLSB = pow(2, src1.chan[n]) – 1;
                        if (maskLSB & src0.chan[n] == 0) {
                                dst.chan[n] = sign(src0.chan[n]) *
                                        ((abs)src0.chan[n] >> src1.chan[n]);
                        } else {
                                dst.chan[n] = sign(src0.chan[n]) *
                                        ((abs)src0.chan[n] >> src1.chan[n])–1;
                        }
                }
        }
}
```

Description:

The *asr* instruction performs component-wise arithmetic right shift of <src0> and storing the results in <dst>. Arithmetic right shift performs sign-extension by repeating the MSB of each data channel of <src0>. The amount of bit shift is provided by <src1>. Only the 5 LSBs of each channel of <src1> are used as an unsigned integer value. The rest of MSBs of <src1> data channels are ignored.

Operands for this instruction can be signed or unsigned integers, but cannot be floating point type. 5-bit shifting applies to packed-dword mode and packed-word mode. For packed word mode, the accumulators have 33 bits per channel.

This instruction is effectively a power-of-2 integer divide with truncate in 2's compliment form. Truncate in 2's compliment form is also known as downward rounding – closest integer that is smaller than or equal to the result. For example, regardless of the bit shift amount in <src1>, the result of arithmetic right-shift of -1 (<src0>) is always -1.

Restrictions:

This instruction does not work with float type operands.

8.3.4 avg – Average

Opcode	Instruction	Description
66 (0x42)	avg <dst> <src0> <src1>	Component-wise averaging of <src0> and <src1> and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[INT]	[INT]

Format:

```
[(<pred>)] avg[.<cmod>] (<exec_size>) <dst> <src0> <src1>
```

Syntax:

```
[(<pred>)] avg[.<cmod>] (<exec_size>) reg reg reg
[(<pred>)] avg[.<cmod>] (<exec_size>) reg reg imm32
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = (src0.chan[n] + src1.chan[n] + 1) >> 1;
        }
}
```

Description:

The *avg* instruction performs component-wise integer average of <src0> and <src1> and stores the results in <dst>. An integer average uses integer upward rounding. It is equivalent to increment one to the addition of <src0> and <src1> and then apply an arithmetic right shift to this intermediate value.

Restrictions:

This instruction only applies to integer data types. The behavior is undefined if any operand is float.

8.3.5 break – Break [DevSNB]

Opcode	Instruction	Description
40 (0x28)	Break <JIP><UIP>	Terminating enabled execution channels and conditionally breaking out from the inner most loop.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•					

Format:

```
[(<pred>)] break (<exec_size>) <JIP> <UIP>
```

Syntax:

```
[(<pred>)] break (<exec_size>) imm16 imm16
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.channel[n] == 1) {
                if (PMask[n] == 1) {
                        PcIP[n] = IP + <UIP>;
                }
                else {
                        PcIP[n] = IP+1;
                }
        }
}
if (PcIP != (IP + 1)) { //all channels
        Jump(IP + <JIP>);
}
```

Description:

The *break* instruction is used to early-out from the inner most loop, or early-out from the inner swtich block.

When used in a loop, upon execution, the *break* instruction terminates the loop for all execution channels enabled by PMask. This is performed by updating the per channel IP to the <UIP>. In case of all the enabled channels hit the *break* instruction, a jump will be performed to the instruction based <JIP>. <JIP> should be the offset to the end of the inner most conditional or loop block, <UIP> should be the offset to the first instruction after the loop block.

When *break* used in a switch block, upon execution, the *break* instruction terminates the current enabled channels for the rest of the switch code block. In case of all the channels hit the *break* instruction, a jump will be performed to the instruction based on <JIP>. <JIP> should be the offset to the first instruction after the switch block. <UIP> should be the instruction after the entire switch block. In case there are multiple *breaks* inside a single case block, the last *break* instruction must be unconditional, and all the previous conditional *break* instructions must have the <JIP> point to the last unconditional *break* instruction. Each case block must end with a *break* instruction.

The following table describes the 2 16-bit instruction pointer offset. Both the <JIP> and <UIP> are signed 16-bit numbers, added to IP pre-increment In binary, <JIP> and <UIP> are at location <src1> and must be of type W (signed word integer).

Bit	Description
31:16	**UIP (Update Target Offset).** This field specifies the jump distance in number of 64bits data chunks if a jump is taken for the channel. Format = S15. Signed integer in 2's compliment
15:0	**JIP (Jump Target Offset).** This field specifies the jump distance in number of 64bits data chunks if a jump is taken for the instruction. Format = S15. Signed integer in 2's compliment

If SPF is ON, none of the PcIP is updated. The UIP must be used to update the execution IP, the JIP is not used in this case.

8.3.6 call – Call [DevSNB]

Opcode	Instruction	Description
44 (0x2c)	call <dst> <JIP>	Subroutine call.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•					DW

Format:

[(<pred>)] call (2) <dst> <JIP>

Syntax:

[(<pred>)] call (2) reg imm16

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < 16; n++) {
        if (WrEn.channel[n] == 1) {
                PcIP[n] = IP + <JIP>;
                CallMask[n] = 1;
        } else {
                PcIP[n] = IP + 1;
                CallMask[n] = 0;
        }
}

if (PcIP[n] != (IP+1)) { // any channel jumped
        dst.chan[0] = IP+1;
        dst.chan[1] = CallMask;
        Jump(IP + <JIP>);
}
```

Description:

The *call* instruction does the jump to the subroutine. It can be predicated or non-predicated. If non-predicated, all enabled channels will jump to the subroutine. If predicated, only the channels enabled by the PMask will jump to the subroutine, the rest of the channels will move to the next instruction after the call instruction, if none of the channels jump into the subroutine, the *call* instruction will be treated as a NOP.

In case of a jump, the *call* instruction stores the return IP onto the first dword of the destination register, the CallMask is stored in the second dword of the destination register.

If SPF is ON, none of the PcIP is updated.

The following table describes the 16-bit jump offset <JIP>. <JIP> must be an immediate operand, whereas it is a signed 16-bit number. When a jump occurs, this value is added to IP pre-increment. In DevSNB instruction binary, <JIP> is at location <src1> and must be of type W (signed word integer).

When SPF is on, the predication control must be scalar.

Bit	Description
31:16	**Reserved:** MBZ
15:0	**JIP (Jump Instruction Count).** This field specifies the jump distance in number of 64bits data chunks if a jump is taken for the instruction. Format = S15. Signed integer in 2's compliment

Restrictions:

- The *call* instruction must have DWord source and destination type, and the destination must be QWord aligned.
- The source0 regioning control must be <2;2,1>, and the execution size must be 2.
- QtrCtrl is not allowed for this instruction.
- **[Errata][DevSNB]** When predication is required for a CALL, the instruction must be invoked inside an "if-endif" block.
- [Errata] [DevSNB-**Astep**] The call instruction does not push the correct callmask onto the stack if it is predicated or resides in a conditional code block.
- [Errata] [DevSNB-**Astep**] The destination must be Oword aligned.

8.3.7 case – Case [DevSNB]

Opcode	Instruction	Description
38 (0x26)	case <dst> <src0> <src1> <JIP>	Signifying the start of an case block of code.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•		•			

Format:

[(<pred>)] case (<exec_size>) null null null <JIP>
case[.<cmod>] (<exec_size>) null <src0> <src1> <JIP>

Syntax:

[(<pred>)] case (<exec_size>) null null null imm16
case[.<cmod>] (<exec_size>) null reg reg imm16
case[.<cmod>] (<exec_size>) null reg imm32 imm16

Pseudocode:

Evaluate(WrEn);

```
for (n = 0; n < 16; n++) {
        if (WrEn.channel[n] == 1) {
                if (<cmod> == 0) { // no embedded compare
                        if (PMask.channel[n] == 0) {
                                PcIP[n] = IP + <JIP>;
                        }
                        else
                        {
                                PcIP[n] = IP + 1;
                        }
                }
                else { // with embedded compare
                        if (cmod.channel[n] == 0) {
                                PcIP[n] = IP + <JIP>;
                        }
                        else
                        {
                                PcIP[n] = IP + 1;
                        }
                }
        }
}
if (<cmod> == 0) { // no embedded compare
        if (PcIP != (IP+1)) { // all channels false
                Jump(IP + <JIP>);
        }
}
else { // with embedded compare
        if (PcIP != (IP+1)) { // all channels false
                Jump(IP + <JIP>);
        }
}
```

Description:

The *case* instruction starts an case/break code block. It restricts execution within the conditional block to only those channels that were enabled via either the predicate control or the condition from <cmod>.

Each *case* instruction must have a matching *break* instruction.

If all channels are inactive (for the case/break block), a jump is performed of the relative distance as specified in the instruction. This jump must be to right after the matching *break* instruction when present, or otherwise to the end of switch code block.

The following table describes the 16-bit exit code <JIP>. <JIP> must be an immediate operand, whereas it is a signed 16-bit number. When a jump occurs, this value is added to IP pre-increment. In DevSNB instruction binary, <JIP> is at location <dst> and must be of type W (signed word integer).

Bit	Description
31:16	**Reserved:** MBZ
15:0	**JIP (Jump Instruction Count).** This field specifies the jump distance in number of 64bits data chunks if a jump is taken for the instruction. Format = S15. Signed integer in 2's compliment

If SPF is ON, none of the PcIP is updated.

Restrictions:

To use embedded compares, the predicate control field for this instruction must be zero, and the conditional modifier field must be none zero. The destination must follow the rules below:

1. Must has the same element size as source0
2. Must have horizontal stride of 1

To use predicated *case* instruction, the conditional modifier field must be zero.

The *case* code block must end with a *break* instruction.

The *case* instruction must be used within the switch/endswitch code block

3. Accumulator can not be used as source or destination for this instruction.

8.3.8 cmp – Compare

Opcode	Instruction	Description
16 (0x10)	cmp.<cmod> <dst> <src0> <src1>	Component-wise comparison of <src0> and <src1> according to conditional modifier in <cmod> and storing the results in flag register in <cmod> and <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•		•	•	[FLT]	[FLT]
•		•	•	[INT]	[INT]

Format:

```
[(<pred>)] cmp[.<cmod>] (<exec_size>) <dst> <src0> <src1>
```

Syntax:

```
[(<pred>)] cmp[.<cmod>] (<exec_size>) reg reg reg
[(<pred>)] cmp[.<cmod>] (<exec_size>) reg reg imm32
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        bitMask[n] = 0;
        if (WrEn.chan[n] == 1) {
                results[n] = src0.chan[n] - src1.chan[n];
                bitMask[n] = Condition(results[n]);
                dst.chan[n][0] = bitMask[n];
        }
}
flag# = bitMask;
```

Description:

The *cmp* instruction performs component-wise comparison of <src0> and <src1> and stores the results in the selected flag register and in <dst>. It takes component-wise subtraction of <src0> and <src1>, evaluating the conditional code (excluding NS signal) based on the conditional modifier, and storing the conditional bits in bit-packed form in the destination flag register and, optionally, in vector form in the LSB of the channels in <dst>. Conditional modifier field cannot be 0000b, i.e., it must be one of the defined conditional modifier codes. Destination operand can be a GRF, an MRF or a null register. If it is not null, for the enabled channels, the LSB of the result in the destination channel contains the flag value for the

channel. The other bits are undefined. When the instruction operates on packed word format, one GRF register may store up to 16 such comparison results. In dword format, one GRF may store up to 8 results. When the register is used later as a vector of Booleans, as only LSB at each channel contains meaning data, software should make sure all higher bits are masked out (e.g. by 'and-ing' an 0x01 constant).

If <exec_size> is equal or less than 8, when '*SecHalf*' option flag is not set, only the lower 8 bits of the selected flag register is updated; otherwise, the higher 8 bits are updated.

When at least one of the source operands is float, the *cmp* instruction obeys the floating point rules detailed in the tables in the Floating Point Mode section of Data Type chapter.

Restrictions:

Destination operand cannot be an ARF register, including accumulator.

Saturation modifier cannot be set in this instruction.

Dword integer source is not allowed for this instruction in float execution mode. In other words, if one source is of type float (:f, :vf), the other source cannot be of type dword integer (:ud or :d).

[DevSNB Errata] The *cmp* instruction can not use accumulator as source if source modifier is used, all other cases are ok.

[DevSNB Errata] Integer compare can not use accumulator as source.

8.3.9 cmpn – Compare NaN

Opcode	Instruction	Description
17 (0x11)	cmpn.<cmod> <dst> <src0> <src1>	Performing component-wise special NaN comparison of <src0> and <src1> according to conditional modifier in <cmod> and storing the results in flag register in <cmod> and <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•		•	•	[FLT]	[FLT]

Format:

[(<pred>)] cmpn[.<cmod>] (<exec_size>) <dst> <src0> <src1>

Syntax:

[(<pred>)] cmpn[.<cmod>] (<exec_size>) reg reg reg
[(<pred>)] cmpn[.<cmod>] (<exec_size>) reg reg imm32

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        bitMask[n] = 0;
        if (EMask.chan[n] == 1) {
                results[n] = src0.chan[n] - src1.chan[n];
                bitMask[n] = ConditionNaN(results[n]);
                dst.chan[n][0] = bitMask[n];
        }
}
flag# = bitMask;
```

Description:

The *cmpn* instruction performs component-wise special-NaN comparison of <src0> and <src1> and stores the results in the selected flag register and in <dst>. It takes component-wise subtraction of <src0> and <src1>, evaluating the conditional signals including NS based on the conditional modifier, and storing the conditional flag bits in bit-packed form in the destination flag register and, optionally, in vector form in the LSB of the channels in <dst>. Conditional modifier field cannot be 0000b, i.e., it must be one of the defined conditional modifier codes. Destination operand can be a GRF, an MRF or a null register. If it is not null, for the enabled channels, the LSB of the result in the destination channel contains the flag value for the channel. The other bits are undefined. When the instruction operates on packed word format, one GRF register may store up to 16 such comparison results. In dword format, one GRF may store up to 8 results. When the register is used later as a vector of Booleans, as only LSB at each channel contains meaning data, software should make sure all higher bits are masked out (e.g. by 'and-ing' an 0x01 constant).

If <exec_size> is equal or less than 8, when '*SecHalf*' option flag is not set, only the lower 8 bits of the selected flag register is updated; otherwise, the higher 8 bits are updated.

When at least one of the source operands is float, the *cmpn* instruction obeys the floating point rules detailed in the tables in the Floating Point Mode section of Data Type chapter.

This instruction is similar to *cmp*. The only difference is that if the second source operand <src1> is a NaN, the result for any conditional modifier except **.nz** is true (see Section 4.3.1.4for details).

For integer operands, *cmpn* and *cmp* are identical.

Restrictions:

Destination operand cannot be an ARF register, including accumulator.

Saturation modifier cannot be set in this instruction.

Dword integer source is not allowed for this instruction in float execution mode. In other words, if one source is of type float (:f, :vf), the other source cannot be of type dword integer (:ud or :d).

DevSNB Errata] Integer compare can not use accumulator as source.

8.3.10 cont – Continue [DevSNB]

Opcode	Instruction	Description
41 (0x29)	cont <JIP><UIP>	Temporally disabling enabled execution channels for the remainder of the inner most loop and conditionally jumping to the last instruction (while) of the loop.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•					

Format:

```
[(<pred>)] cont (<exec_size>) <JIP> <UIP>
```

Syntax:

```
[(<pred>)] cont (<exec_size>) imm16 imm16
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < 16; n++) {
        if (WrEn.channel[n] == 1) {
                if (PMask[n] == 1) {
                        PcIP[n] = IP + <UIP>;
                }
                else {
                        PcIP[n] = IP + 1;
                }
        }
}
if (PcIP != (IP+1)) { //all channel true
        Jump(IP + <JIP>);
}
```

Description:

The *cont* instruction disables execution for the subset of channels for the remainder of the current loop iteration. Channels remain disabled until right before the *while* instuction or right before the condition check code block for the *while* instruction. In case of all enabled channels hit this instruction, a jump is made a distance of <JIP> where execution continues.

The following table describes the two 16-bit exit code, <JIP> and <UIP>. The field are signed 16-bit numbers, added to IP pre-increment. The <UIP> should always point to the loop's associated 'while' instruction, and the <JIP> should point to the last instruction of the inner most conditional block if the *cont* instruction is inside a conditional block. In case of the *break* instruction directly under the loop, the <JIP> and the <UIP> will be the same. In DevSNB binary, <JIP> and <UIP> are at location <src1> and must be of type W (signed eword integer).

Bit	Description
31:16	**UIP (Update Target Offset).** This field specifies the jump distance in number of 64bits data chunks if a jump is taken for the channel. Format = S15. Signed integer in 2's compliment
15:0	**JIP (Jump Target Offset).** This field specifies the jump distance in number of 64bits data chunks if a jump is taken for the instruction. Format = S15. Signed integer in 2's compliment

If SPF is ON, none of the PcIP is updated. The UIP must be used to update the execution IP, the JIP is not used in this case.

(1) PMask here is for all the channels enabled for the *cont* instruction.

8.3.11 dp2 – Dot Product 2

Opcode	Instruction	Description
87 (0x57)	Dp2 <dst> <src0> <src1>	Performing two-wide dot product in four-tuples of <src0> and <src1> and storing the replicated results in four-tuples in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[FLT]	[FLT]

Format:

```
[(<pred>)] dp2[.<cmod>] (<exec_size>) <dst> <src0> <src1>
```

Syntax:

```
[(<pred>)] dp2[.<cmod>] (<exec_size>) reg reg reg
[(<pred>)] dp2[.<cmod>] (<exec_size>) reg reg imm32
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n+=4) {
        fTmp = src0.chan[n] * src1.chan[n]
                + src0.chan[n+1] * src1.chan[n+1];
```

```
        if (WrEn.chan[n] == 1) dst.chan[n] = fTmp;
        if (WrEn.chan[n+1] == 1) dst.chan[n+1] = fTmp;
        if (WrEn.chan[n+2] == 1) dst.chan[n+2] = fTmp;
        if (WrEn.chan[n+3] == 1) dst.chan[n+3] = fTmp;
    }
```

Description:

The *dp2* instruction performs a two-wide dot-product on four-tuple vector basis and storing the same scalar result per four-tuple to all four channels in <dst>. This instruction is similar to dp4 except that every third and fourth elements of <src0> (post-source-swizzle if present) are not involved in the computation.

Special care has been taken in the hardware such that if the resulting value for a given group of four channels is 0.0f, the sign of the result correctly reflects the input data of the first two channels of the group of four.

Restrictions:

Source operands cannot be an accumulator register.

Execution size cannot be less than 4.

This instruction does not support integer operation.

Horizontal stride must be 1.

[Pre-DevSNB] The results are NOT stored in the accumulator register. This instruction does implicitely update accumulator register, however with undefined values.

[DevSNB+] The results can be stored in the accumulator register, the channels are wirrten the same way as the <dst> register with replication.

8.3.12 dp3 – Dot Product 3

Opcode	Instruction	Description
86 (0x56)	dp3 <dst> <src0> <src1>	Performing three-wide dot product in four-tuples of <src0> and <src1> and storing the replicated results in four-tuples in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[FLT]	[FLT]

Format:

```
[(<pred>)] dp3[.<cmod>] (<exec_size>) <dst> <src0> <src1>
```

Syntax:

```
[(<pred>)] dp3[.<cmod>] (<exec_size>) reg reg reg
[(<pred>)] dp3[.<cmod>] (<exec_size>) reg reg imm32
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n+=4) {
        fTmp = src0.chan[n] * src1.chan[n]
                + src0.chan[n+1] * src1.chan[n+1]
                + src0.chan[n+2] * src1.chan[n+2];
        if (WrEn.chan[n] == 1) dst.chan[n] = fTmp;
        if (WrEn.chan[n+1] == 1) dst.chan[n+1] = fTmp;
        if (WrEn.chan[n+2] == 1) dst.chan[n+2] = fTmp;
        if (WrEn.chan[n+3] == 1) dst.chan[n+3] = fTmp;
}
```

Description:

The *dp3* instruction performs a three-wide dot-product on four-tuple vector basis and storing the same scalar result per four-tuple to all four channels in <dst>. This instruction is similar to dp4 except that every fourth element of <src0> (post-source-swizzle if present) is not involved in the computation.

Special care has been taken in the hardware such that if the resulting value for a given group of 4 channels is 0.0f, the sign of the result correctly reflects the input data of the first three channels of the group of four.

Restrictions:

Source operands cannot be an accumulator register.

Execution size cannot be less than 4.

This instruction does not support integer operation.

Horizontal stride must be 1.

[Pre-DevSNB]The results are NOT stored in the accumulator register. This instruction does implicitely update accumulator register, however with undefined values.

[DevSNB+]The results can be stored in the accumulator register, the channels are wirrten the same way as the <dst> register with replication.

8.3.13 dp4 – Dot Product 4

Opcode	Instruction	Description
84 (0x54)	dp4 <dst> <src0> <src1>	Performing four-wide dot product of <src0> and <src1> and storing the four-wide replicated results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[FLT]	[FLT]

Format:

[(<pred>)] dp4[.<cmod>] (<exec_size>) <dst> <src0> <src1>

Syntax:

[(<pred>)] dp4[.<cmod>] (<exec_size>) reg reg reg
[(<pred>)] dp4[.<cmod>] (<exec_size>) reg reg imm32

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n+=4) {
        fTmp = src0.chan[n] * src1.chan[n]
                + src0.chan[n+1] * src1.chan[n+1]
                + src0.chan[n+2] * src1.chan[n+2]
                + src0.chan[n+3] * src1.chan[n+3];
        if (WrEn.chan[n] == 1) dst.chan[n] = fTmp;
        if (WrEn.chan[n+1] == 1) dst.chan[n+1] = fTmp;
        if (WrEn.chan[n+2] == 1) dst.chan[n+2] = fTmp;
        if (WrEn.chan[n+3] == 1) dst.chan[n+3] = fTmp;
}
```

Description:

The *dp4* instruction performs a four-wide dot-product on four-tuple vector basis and storing the same scalar result per four-tuple to all four channels in <dst>.

Restrictions:

Source operands cannot be an accumulator register.

Execution size cannot be less than 4.

This instruction does not support integer operation.

Horizontal stride must be 1.

[Pre-DevSNB]The results are NOT stored in the accumulator register. This instruction does implicitely update accumulator register, however with undefined values.

[DevSNB+]The results can be stored in the accumulator register, the channels are wirrten the same way as the <dst> register with replication.

8.3.14 dph –Dot Product Homogeneous

Opcode	Instruction	Description
85 (0x55)	dph <dst> <src0> <src1>	Performing four-wide homogeneous dot product of <src0> and <src1> and storing the four-wide replicated results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[FLT]	[FLT]

Format:

```
[(<pred>)] dph[.<cmod>] (<exec_size>) <dst> <src0> <src1>
```

Syntax:

```
[(<pred>)] dph[.<cmod>] (<exec_size>) reg reg reg
[(<pred>)] dph[.<cmod>] (<exec_size>) reg reg imm32
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n+=4) {
        fTmp = src0.chan[n] * src1.chan[n]
                + src0.chan[n+1] * src1.chan[n+1]
                + src0.chan[n+2] * src1.chan[n+2]
                + src1.chan[n+3];
        if (WrEn.chan[n] == 1) dst.chan[n] = fTmp;
        if (WrEn.chan[n+1] == 1) dst.chan[n+1] = fTmp;
        if (WrEn.chan[n+2] == 1) dst.chan[n+2] = fTmp;
        if (WrEn.chan[n+3] == 1) dst.chan[n+3] = fTmp;
}
```

Description:

The *dph* instruction performs a four-wide homogeneous dot-product on four-tuple vector basis and storing the same scalar result per four-tuple to all four channels in <dst>. This instruction is similar to dp4 except that every fourth element of <src0> (post-source-swizzle if present) is forced to 1.0f.

Restrictions:

Source operands cannot be an accumulator register.

Execution size cannot be less than 4.

This instruction does not support integer operation.

Horizontal stride must be 1.

[Pre-DevSNB]The results are NOT stored in the accumulator register. This instruction does implicitely update accumulator register, however with undefined values.

[DevSNB+]The results can be stored in the accumulator register, the channels are wirrten the same way as the <dst> register with replication.

8.3.15 else – Else [DevSNB]

Opcode	Instruction	Description
36 (0x24)	else <JIP>	An optional statement within an if/else/endif block of code.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types

Format:

```
else (<exec_size>) <JIP>
```

Syntax:

```
else (<exec_size>) imm16
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < 16; n++) {
        if (WrEn.channel[n] == 1) {
                PcIP[n] = IP + <JIP>;
        }
}
```

```
if (PcIP != (IP+1)) { // for all channels
        Jump(IP + <JIP>);
}
```

Description:

The *else* instruction is an optional statement within an *if/else/endif* block of code. It restricts execution within the else/endif portion to the opposite set of channels enabled under the if/else portion. Channels which were inactive prior to entering the *if/endif* block remain inactive throughout the entire block.

All enabled channels upon arriving the *else* instruction will be redirected to the matching *endif*. If all channels are redirected (by *else* or before *else*), a relative jump is performed to the location specified by <JIP>. The jump target should be the the matching *endif* instruction for that conditional block.

The following table describes the 16-bit <JIP>. In DevSNB binary, <JIP> is at location <dst> and must be of type W (signed word integer). <JIP> must be an immediate operand, it is a signed 16-bit number and is intended to be forward referencing. This value is added to IP pre-increment.

Bit	Description
31:16	**Reserved:** MBZ
15:0	**JIP (Jump Instruction Count).** This field specifies the jump distance in number of 64bits data chunks if a jump is taken for the instruction. Format = S15. Signed integer in 2's compliment

If SPF is ON, none of the PcIP is updated.

Restrictions:

Predication is not allowed.

8.3.16 endif – End-If

Opcode	Instruction	Description
37 (0x24)	endif <JIP>	Restoring execution to those data channels that were active prior to the if/else/endif block. Jump to next hop point if all channels are disabled.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types

Format:

endif <JIP>

Syntax:

endif imm16

Pseudocode:

```
Evaluate(WrEn);
if (WrEn == 0) { // all channels false
        Jump(IP + <JIP>);
}
```

Description:

The *endif* instruction terminates an if/else/endif block of code. It restores the execution to these data channels that were active prior to the if/else/endif block.

The *endif* instruction is also used to hop out nested conditionals by jumping to the end of the next outer conditional block when all channels are disabled.

The following table describes the 16-bit jump target offset <JIP>. In DevSNB binary, <JIP> is at location <src1> and must be of type W (signed word integer). <JIP> must be an immediate operand.

Bit	Description
31:16	**Reserved:** MBZ
15:0	**JIP (Jump Target Offset).** This field specifies the jump distance in number of 64bits data chunks if a jump is taken for the instruction. Format = S15. Signed integer in 2's compliment

Execution size is ignored for this instruction.

If SPF is ON, none of the PcIP is updated.

Restrictions:

Predication is not allowed.

Accumulator can not be used as source or destination for this instruction.

8.3.17 fork – Fork [DevSNB]

Opcode	Instruction	Description
46 (0x2E)	fork <src0> <JIP>	Set active set of 16 channels according to <src0>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•					

Format:

[(<pred>)] fork (<exec_size>) <src0> <JIP>

Syntax:

[(<pred>)] fork (<exec_size>) reg imm16

Pseudocode:

```
If (src0 == NULL) {
        SPF = 1;
} else {
        SPF = 0;

        for (n = 0; n < exec_size; n++) {
                if (src0 == DMask) {
                        if (DMask.channel[n] == 0) {
                                PcIP[n] = IP + <JIP>;
                        }
                        else {
                                PcIP[n] = IP+1;
                        }
                }
                if (src0 == VMask) {
                        if (VMask.channel[n] == 0) {
                                PcIP[n] = IP + <JIP>;
                        }
                        else {
                                PcIP[n] = IP+1;
                        }
                }
```

```
        }

        if (PcIP != (IP+1)) { // for all channels
                Jump(IP + <JIP>);
        }
}
```

Description:

The *fork* instruction starts a *fork* block of code. It restricts execution within the *fork* block to only those channels that were enabled via select register (dispatch mask or vector mask) and further qualified by the predicate control.

If all channels are inactive (for the *fork* block), a jump is performed of the relative distance as specified in the instruction.

The following table describes the 16-bit exit code <JIP>. <JIP> must be an immediate operand, whereas JumpIP is a signed 16-bit number. When a jump occurs, this value is added to IP pre-increment. In DevSNB instruction binary, <JIP> is at location <dst> and must be of type W (signed word integer).

If the register specified is NULL, fork simply set SPF = 1.

The *fork* instruction should be used to turn SPF off in a middle of a program, it will bring the PcIPs the the current ExIP for continuing execution.

Bit	Description
15:0	**JIP (Jump Instruction Count).** This field specifies the jump distance in number of 64bits data chunks if a jump is taken for the instruction. Format = S15. Signed integer in 2's compliment

Restrictions:

- Instruction is only allowed at global scope.
- Src0 is limited to DMask.0, Dmask.1, Vmask.0, Vmask.1 and NULL.
- The *fork* instruction should only be used to enable 32 channel execution where per-channel branch is required. The SPF control must be set during thread dispatch time based on 32 channel dispatch. Within the shader, the fork instruction should only be used to switch between the 3 modes below:
  - 32 channel execution in SPF on mode, where no per-channel branch can be performed
  - lo-16 channel execution in SPF off mode, where per-channel branch on the low 16 channels can be performed, **QtrCtrl {H1} must be used in this case.**

- hi-16 channel execution in SPF off mode, where per-channel branch on the high 16 channels can be performed, **QtrCtrl {H2} must be used in this case.**

- Qtr control is not allowed for the *fork* instruction.

8.3.18 frc – Fraction

Opcode	Instruction	Description
67 (0x43)	frc <dst> <src0>	Taking component-wise truncate-to-minus-infinity fraction operation of <src0> and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•		•	•	[FLT]	[FLT]
•		•	•	[FLT]	[INT]

Format:

[(<pred>)] frc[.<cmod>] (<exec_size>) <dst> <src0>

Syntax:

[(<pred>)] frc[.<cmod>] (<exec_size>) reg reg
[(<pred>)] frc[.<cmod>] (<exec_size>) reg imm32

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = src0.chan[n] – floor(src0.chan[n]);
        }
}
```

Description:

The *frc* instruction computes, component-wise, the truncate-to-minus-infinity fractional values of <src0> and stores the results in <dst>. The results, in the range of [0.0, 1.0], are the fractional portion of the source data.

Source operand for this instruction must be of floating point type. This instruction can only operate on normalized floating source and therefore can not take accumulator as source or destination operand.

This instruction only applies to floating point operands.

Floating point fraction computation follows rules in Table 8-3 (or Table 8-4), if the current floating point mode is IEEE mode (or ALT mode).

Table 8-3. Floating point fraction computation in IEEE mode

<src0>	**–inf**	**–finite**	**–denorm**	**–0**	**+0**	**+denorm**	**+finite**	**+inf**	**NaN**
<dst>	NaN	*	+0	+0	+0	+0	*	NaN	NaN

Notes:

* Result is in the range of [+0, 1) – not including 1.

Table 8-4. Floating point fraction computation in ALT mode

<src0>	**– fmax**	**–finite**	**–denorm**	**–0**	**+0**	**+denorm**	**+finite**	**+ fmax**	******
<dst>	+0	*	+0	+0	+0	+0	*	+0	

Notes:

* Result is in the range of [+0, 1) – not including 1.

** Result is undefined if <src0> is {–inf, +inf, NaN}.

Restrictions:

Saturation modifier does not apply to this instruction.

This instruction can not take accumulator as source or destination operand as it can only operate on normalized floating source.

This instruction does implicitely update accumulator register when enabled, however with undefined values.

8.3.19 halt – Halt [DevSNB]

Opcode	Instruction	Description
42 (0x2A)	halt <JIP> <UIP>	Temporarily suspending execution for all enabled execution channels.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•					

Format:

[(<pred>)] halt (<exec_size>) <JIP> <UIP>

Syntax:

[(<pred>)] halt (<exec_size>) imm16 imm16

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < 16; n++) {
        if (WrEn.channel[n] == 1) {
                if (PMask.channel[n] == 1) {
                        PcIP[n] = IP + <UIP>;
                else {
                        PcIP[n] = (IP+1);
                }
        }
}
if (PcIP != (IP+1)) { // for all channels
        Jump(IP + <JIP>);
}
```

Description:

The *halt* instruction temporarily suspends execution for all enabled compute channels. Upon execution, the enabled channels will be sent to the instruction at (IP+<UIP>), if all channels are enabled at HALT, a jump to (IP+<JIP>) will be performed.

In case of the *halt* instruction not inside any conditional code block, the value of <JIP> and <UIP> should be the same. In case of the *halt* instruction inside conditional code block, the <UIP> should be the end of the program, and the <JIP> should be end of the most inner conditional code block.

The following table describes the 16-bit jump offsets. In DevSNB binary, <JIP> and <UIP> are at location <src1> and must be of type W (signed word integer). The <JIP> and <UIP> are added to IP pre-increment.

Bit	Description
31:16	**UIP (Update Target Offset).** This field specifies the jump distance in number of 64bits data chunks if a jump is taken for the channel. Format = S15. Signed integer in 2's compliment
15:0	**JIP (Jump Target Offset).** This field specifies the jump distance in number of 64bits data chunks if a jump is taken for the instruction. Format = S15. Signed integer in 2's compliment

If SPF is ON, none of the PcIP is updated. The UIP must be used to update the execution IP, the JIP is not used in this case.

Restrictions:

<dst> and <src0> must be NULL.

8.3.20 if – If [DevSNB]

Opcode	Instruction	Description
34 (0x22)	if <dst> <src0> <src1> <JIP>	Signifying the start of an if/else/endif block of code.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•		•			

Format:

```
[(<pred>)] if (<exec_size>) null null null <JIP>
        if[.<cmod>] (<exec_size>) null <src0> <src1> <JIP>
```

Syntax:

```
[(<pred>)] if (<exec_size>) null null null imm16
        if[.<cmod>] (<exec_size>) null reg reg imm16
        if[.<cmod>] (<exec_size>) null reg imm32 imm16
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < 16; n++) {
        if (WrEn.channel[n] == 1) {
```

```
                if (<cmod> == 0) { // no embedded compare
                        if (PMask.channel[n] == 0) {
                                PcIP[n] = IP + <JIP>;
                        }
                        else {
                                PcIP[n] = IP+1;
                        }
                }
                else { // with embedded compare
                        if (cmod.channel[n] == 0) {
                                PcIP[n] = IP + <JIP>;
                        }
                        else {
                                PcIP[n] = (IP+1);
                        }
                }
        }
}
if (<cmod> == 0) { // no embedded compare
        if (PcIP != (IP+1)) { // for all channels
                Jump(IP + <JIP>);
        }
}
else { // with embedded compare
        if (PcIP != (IP+1)) { // for all channels
                Jump(IP + <JIP>);
        }
}
```

Description:

The *if* instruction starts an if/endif or an if/else/endif block of code. It restricts execution within the conditional block to only those channels that were enabled via either the predicate control or the condition from <cmod>.

Each *if* instruction must have a matching *endif* instruction and may have up to one matching *else* instruction before *endif*.

If all channels are inactive (for the if/endif or if/else block), a jump is performed of the relative distance as specified in the instruction. This jump must be to right after the matching *else* instruction when present, or otherwise to the matching *endif* instruction of that conditional block.

The following table describes the 16-bit exit code <JIP>. <JIP> must be an immediate operand, whereas JumpIP is a signed 16-bit number. When a jump occurs, this value is added to IP pre-increment. In DevSNB instruction binary, <JIP> is at location <dst> and must be of type W (signed word integer).

Bit	Description
15:0	**JIP (Jump Instruction Count).** This field specifies the jump distance in number of 64bits data chunks if a jump is taken for the instruction. Format = S15. Signed integer in 2's compliment

If SPF is ON, none of the PcIP is updated.

Restrictions:

To use embedded compares, the predicate control field for this instruction must be zero, and the conditional modifier field must be none zero. The destination must follow the rules below:

3. Must has the same element size as source0
4. Must have horizontal stride of 1

To use predicated *if* instruction, the conditional modifier field must be zero.

8.3.21 jmpi – Jump Indexed

Opcode	Instruction	Description
32 (0x20)	jmpi <index>	Redirecting program execution to <index> instructions forward of the current post-incremented instruction pointer.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•					

Format:

```
[(<pred>)] jmpi (1) <exitcode> {NoMask}
```

Syntax:

```
[(<pred>)] jmpi (1) reg32 {NoMask}
[(<pred>)] jmpi (1) imm32 {NoMask}
```

Pseudocode:

```
Evaluate(WrEn);
if (WrEn != 0) {
        Jump(<exitcode.index> + 1); // +1 if compacted, +2 if not.
}
```

Description:

The *jmpi* instruction redirects program execution to <exitcode.index> instructions forward of the current **post-incremented** instruction pointer. <exitcode.index> is treated a signed integer value, with positive integers or zero generating forward jumps, and negative integers generating backward jumps. An <exitcode.index> value of 0 means execution continues at the instruction immediately following the *jmpi* instruction, while an index value of -1 would imply an infinite loop.

Bit	Description
31:0	**index (Jump Index)** This field specifies byte aligned jump distance if a jump is taken for the instruction. Format = S31. Signed integer in 2's compliment

<exitcode> may be a scalar register or an immediate. The data type of <exitcode > must be D (signed doubleword integer).

This instruction executes regardless of the calculated WrEn at the time of issue. – To reduce hardware complexity, instruction optional control {NoMask} must be set for this instruction. This instruction invokes a thread switch after issue to allow any masks and/or IP to be resolved if necessary.

Execution size must be 1.

Predication is allowed to provide conditional jump with a scalar condition. As the execution size is 1, the first channel of PMASK (flags post prediction control and negate) is used to determine whether the jump is taken or not. If the condition is false, the jump is not taken and the IP immediately following will be executed next.

In DevSNB binary, <exitcode.index> is at location <src1>. IP register must be put (for example, by the assembler) at <dst> and <src0> locations.

If SPF is ON, none of the PcIP is updated

8.3.22 line – Line

Opcode	Instruction	Description
89 (0x59)	Line <dst> <src0> <src1>	Computing a component-wise line equation (v = p*u+q) of <src0> and <src1> and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[FLT]	[FLT]
•	•	•	•	[INT]	[INT]

Format:

```
[(<pred>)] line[.<cmod>] (<exec_size>) <dst> <src0> <src1>
```

Syntax:

```
[(<pred>)] line[.<cmod>] (<exec_size>) reg reg reg
[(<pred>)] line[.<cmod>] (<exec_size>) reg reg imm32
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
for (n = 0; n < exec_size; n++) {
        dwP = src0.RegNum.SubRegNum[bits4:2]   // a DW aligned scalar
        dwQ = src0.RegNum.(SubRegNum[bit4]|0x8)        // 4-th component
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = dwP * src1.chan[n] + dwQ
        }
}
```

Description:

The *line* instruction computes a component-wise line equation (*v* = *p***u*+*q* where *u*/*v* are vectors and *p*/*q* are scalars) of <src0> and <src1> and storing the results in <dst>. <src1> is the input vector *u*. <src0> provides input scalars *p* and *q*, where *p* is the scalar value based on the region description of <src0> and *q* is the scalar value implied from <src0> region. Specifically, *q* is the fourth component of the 4-tuple (128-bit aligned) that *p* belongs to.

Restrictions:

This is a specialized instruction that only support execution size of 8 or 16.

<src0> region must be a replicated scalar (with HorzStride = VertStride = 0).

Dword integer source is not allowed for this instruction in float execution mode. In other words, if one source is of type float (:f, :vf), the other source cannot be of type dword integer (:ud or :d).

Source operands cannot be an accumulator register.

<src0> for *line* instruction has to have .0 or .4 as the subregister number.

8.3.23 lzd – Leading Zero Detection

Opcode	Instruction	Description
74 (0x4A)	lzd <dst> <src0>	Performing component-wise leading zero detection of <src0> and storing the results in <dst>.

Instr Comp	Imp'd Accu	Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	•		[INT]	[INT]

Format:

```
[(<pred>)] lzd[.<cmod>] (<exec_size>) <dst> <src0>
```

Syntax:

```
[(<pred>)] lzd[.<cmod>] (<exec_size>) reg reg
[(<pred>)] lzd[.<cmod>] (<exec_size>) reg reg
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                UD udScalar = src0.chan[n];
                UD cnt = 0;
                while ( (udScalar & (1<<31)) == 0 && cnt != 32) {
                        cnt ++;
                        udScalar = udScalar << 1;
                }
                dst.chan[n] = cnt;
        }
}
```

Description:

The *lzd* instruction counts component-wise the leading zeros from <src0> and storing the resulting counts in <dst>.

This instruction only work on unsigned dword source. Source operand may be a signed or unsigned. If it is a signed integer, source modifier (abs) must be used to convert the source into an unsigned integer type.

The destination operand must also be of unsigned dword type.

Restrictions:

The destination operand cannot be the accumulator.

This instruction does implicitely update accumulator register when enabled, however with undefined values.

8.3.24 lrp – Linear Interpolation

Opcode	Instruction	Description
92 (0x5c)	lrp <dst> <src0> <src1>	Computing a component-wise lrp equation (w = u*x+v*(1-x) of vectors (x, u, v) from <src0>, <src1>, and <src2> and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•		•	•	[FLT]	[FLT]

Format:

```
[(<pred>)] lrp[.<cmod>] (<exec_size>) <dst> <src0> <src1> <src2>
```

Syntax:

```
[(<pred>)] lrp[.<cmod>] (<exec_size>) reg reg reg
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = src1.chan[n] * src0.chan[n] + src2.chan[n] * (1 –
src0.chan[n])
        }
}
```

Description:

The *lrp* instruction takes component-wise multiplication of <src0> and <src1>, and adds the result to the component-wise multiplication of <src2> and (1 - <src0>) , and then stores the final results in <dst>.

The *lrp* instruction uses the 3-source instruction format.

Restrictions:

This instruction only supports float source and destination.

Immediate source is not allowed for *lrp*.

The vertical stride is overloaded to 4 in HW for 3-src instructions.

The overflow conditional modifier is not allowed.

8.3.25 mac – Multiply Accumulate

Opcode	Instruction	Description
72 (0x48)	mac <dst> <src0> <src1>	Performing component-wise multiply accumulate of <src0> and <src1> and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[FLT]	[FLT]
•	•	•	•	[INT]	[INT]

Format:

```
[(<pred>)] mac[.<cmod>] (<exec_size>) <dst> <src0> <src1>
```

Syntax:

```
[(<pred>)] mac[.<cmod>] (<exec_size>) reg reg reg
[(<pred>)] mac[.<cmod>] (<exec_size>) reg reg imm32
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = src0.chan[n] * src1.chan[n] + acc0.chan[n]
        }
}
```

Description:

The *mac* instruction takes component-wise multiplication of <src0> and <src1>, adds the results with the corresponding accumulator values, and then stores the final results in <dst>.

Restrictions:

Accumulator is an implied source to the addition portion of the computation. Explicit source operands cannot be accumulator.

This instruction doesn't support dword integers (D or UD).

8.3.26 mach – Multiply Accumulate High

Opcode	Instruction	Description
73 (0x49)	mach <dst> <src0> <src1>	Performing component-wise multiply accumulation of <src0>, <src1> and accumulator register, and returning the high dword of results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
□			□	[INT]	[INT]

Format:

[(<pred>)] mach[.<cmod>] (<exec_size>) <dst> <src0> <src1>

Syntax:

[(<pred>)] mach[.<cmod>] (<exec_size>) reg reg reg

[(<pred>)] mach[.<cmod>] (<exec_size>) reg reg imm32

Pseudocode:

```
Evaluate(WrEn);

for (n = 0; n < exec_size; n++) {

acc0.chan[n][63:0]

= (src0.chan[n][31:16] * src1.chan[n][31:0])<<16

+ acc0.chan[n][63:0];

if (WrEn.channel[n] == 1) {

dst.channel[n][31:0] = acc0.chan[n][63:32]

}

}
```

Description:

The mach instruction performs dword integer multiply-accumulate operation and outputs the high dword (bits [63:32]). On a component by component basis, this instruction multiplies dwords in <src1> with the high words of dwords in <src0>, left-shifts the results by 16 bits, adds them with the corresponding accumulator values, and keeps the whole 64-bit results in the accumulator. It then stores the high dword (bits [63:32]) of the results in <dst>.

This instruction is intended to be used to emulate 32-bit dword integer multiplication by utilizing the large number of bits available in the accumulator. For example, the following four instructions perform vector multiplication of two 32-bit signed integer source from r2 and r3 and store the resulting vectors with high 32-bit in r5 and low 32-bit in r6.

```
mul  (8) acc0:d      r2.0<8;8,1>d    r3.0<8;8,1>:d

mach (8) rTemp<1>:d r2.0<8;8,1>d    r3.0<8;8,1>:d

mov  (8) r5.0<1>:d    rTemp:d // hi-32bits

mov  (8) r6.0<1>:d   acc0:d // lo-32bits
```

The MUL and MACH instruction should have all channels enabled. The first MOV should have channel enable from the destHI of IMUL, the second MOV should have the channel enable from the destLO of IMUL.

As mach is used to generate part of 64-bit dword integer results, saturation modifier should not be used. In fact, saturation modifier should not be used for any of these four instructions.

Source and destination operands must be dword integers. Source and destination must be of the same type, signed integer or unsigned integer.

If <dst> is UD, <src0> and <src1> may be UD and/or D. However, if any of <src0> and <src1> is D, source modifier, (abs), must be present to convert it to match with <dst>.

If <dst> is D, <src0> and <src1> must also be D. They cannot be UD as it may cause unexpected overflow because the computed results are limited to 64 bits.

Restrictions:

Accumulator is an implied source to the addition portion of the computation. Therefore, explicit source operands cannot be accumulator.

The AccWrEn must be set for this instruction.

The AccWrCtrl control must be set for this instruction.

8.3.27 mad – Multiply Add

Opcode	Instruction	Description
91 (0x5B)	mad <dst> <src0> <src1> <src2>	Component-wise multiply add of <src0>, <src1>, and <src2> and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[FLT]	[FLT]

Format:

[(<pred>)] mad[.<cmod>] (<exec_size>) <dst> <src0> <src1> <src2>

Syntax:

[(<pred>)] mad[.<cmod>] (<exec_size>) reg reg reg reg

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = src1.chan[n] * src2.chan[n] + src0.chan[n]
        }
}
```

Description:

The *mad* instruction follows the 3-src instruction format.

The *mad* instruction takes component-wise multiplication of <src1> and <src2>, adds the results with the corresponding <src0> values, and then stores the final results in <dst>.

Restrictions:

This instruction only supports float source and destination.

Immediate source is not allowed for *mad*.

The vertical stride is overloaded to 4 in HW for 3-src instructions.

.

8.3.28 math – Extended Math Function [DevSNB]

Opcode	Instruction	Description
56 (0x38)	math <dst> <src0> <src1> <FC>	Component-wise extended math function on <src0> (and <src1>) to <dst> based on the <FC>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•			[FLT] [INT]	[FLT] [INT]

Format:

[(<pred>)] math (<exec_size>) <dst> <src0> <src1> <FC>

Syntax:

[(<pred>)] math (<exec_size>) reg reg reg imm6

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.channel[n] == 1) {
                switch FC[3:0] {
                        case 1h:
                                dst.channel[n] = rcp(src0.channel[n]);
                        case 2h:
                                dst.channel[n] = log(src0.channel[n]);
                        case 3h:
                                dst.channel[n] = exp(src0.channel[n]);
                        case 4h:
                                dst.channel[n] = sqrt(src0.channel[n]);
                        case 5h:
                                dst.channel[n] = rsq(src0.channel[n]);
                        case 6h:
                                dst.channel[n] = sin(src0.channel[n]);
                        case 7h:
                                dst.channel[n] = cos(src0.channel[n]);
                        case 9h:
                                // src0 / src1
```

```
                        dst.channel[n] = fdiv(src0.channel[n], src1.channel[n]);
                    case Ah:
                        dst.channel[n] = pow(src0.channel[n], src1/channel[n]);
                    case Bh:
                        // src0 / src1
                        idiv(src0.channel[n], src1.channel[n]);
                        dst.channel[n] = quotient;
                        dst+1.channel[n] = remainder;
                    case Ch:
                        idiv(src0.channel[n], src1.channel[n]);
                        dst.channel[n] = quotient;
                    case Dh:
                        idiv(src0.channel[n], src1.channel[n]);
                        dst.channel[n] = remainder;
                }
                endswitch;
            }
        }
```

Description:

The *math* instruction performs extended math function on the components in <src0>, or <src0> and <src1>, and write the output to the channels of <dst>. The type of extended math function are based on the FC[3:0] encoding in the table below.

Function Control[5:0]	Description
[5]	Reserved
[4]	Reserved
[3:0]	Function Description: 0h: Reserved 1h: INV (reciprocal) 2h: LOG 3h: EXP 4h: SQRT 5h: RSQ

Function Control[5:0]	Description
	6h: SIN 7h: COS 8h: Reserved 9h: FDIV Ah: POW Bh: INT DIV – return quotient and remainder Ch: INT DIV – return quotient only Dh: INT DIV – return remainder Eh: Reserved Fh: Reserved

Restrictions:

The *math* instruction can only use GRF registers as source(s) and destination.

The supported regioning modes for *math* instruction are align16, align1 with the following restrictions:

- Scalar source is supported.
- Source and destination horizontal stride must be 1.
- Width must be the same as execution size.
- Source and destination offset must be the same, except the case of scalar source.

The *math* instruction does not support indirect addressing modes.

For INTDIV with both quotient and remainder, the second destination register is implied from the first destination register with “+1” to get the register number.

DW and UD is the only source format supported for INT DIV, float32 is the only source format supported for all the other functions.

Mixed DW and UD sources are not allowed for the INT DIV function.

For single source math function, <src1> must be programmed as ARF-NULL register.

The *math* instruction can not be SIMD16.

The FDIV function is not supported in ALT_MODE.

The only supported rounding mode for *math* instruction is Round to Nearest Even.

8.3.29 mov – Move

Opcode	Instruction	Description
1 (0x01)	mov <dst> <src0>	Component-wise move from <src0> to <dst>.

Instr Comp	Imp'd Accu	Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•		•	•	•	•	[FLT] [INT]	[FLT] [INT]

Format:

```
[(<pred>)] mov[.<cmod>] (<exec_size>) <dst> <src0>
```

Syntax:

```
[(<pred>)] mov[.<cmod>] (<exec_size>) reg reg
[(<pred>)] mov[.<cmod>] (<exec_size>) reg imm32
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = src0.chan[n]
        }
}
```

Description:

The *mov* instruction moves the components in <src0> into the channels of <dst>. If <src0> and <dst> are of different types, format conversion is performed. If <src0> is a scalar immediate, the immediate value is loaded into all channels of <dst>.

Restrictions:

The accumulator can be either <src0> or <dst> but not both.

This instruction does not implicitely update accumulator register.

Conditional Modifier is not allowed for this instruction.

Raw MOV is not supported for floats with ALT mode if INF/NAN is expected.

A MOV operation with NO source or destination modifiers (srcmod, saturate) or datatype conversion is considered a Raw Mov.

8.3.30 movi – Move Indexed

Opcode	Instruction	Description
3 (0x3)	movi <dst> <src0>	Fast component-wise indexed move from <src0> to <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[FLT] [INT]	[FLT] [INT]

Format:

```
[(<pred>)] movi[.<cmod>] (<exec_size>) <dst> <src0>
```

Syntax:

```
[(<pred>)] movi[.<cmod>] (<exec_size>) reg reg
```

Pseudocode:

```
Evaluate(WrEn)
srcregfile = regfile(src0)
srcreg = reg(address[0])
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                srcsubreg = subreg(address[n] + addr_imm)
                dst.chan[n] = srcregfile.srcreg.srcsubreg
        }
}
```

Description:

The *movi* instruction performs a fast component-wise indexed move for subfields from <src0> to <dst>. The source operand must be an indirectly-addressed register. All channels of the source operand share the same register number, which is provided by the register field of the first address subregister, with a possible immediate address offset. The register fields of the subsequent address subregisters are ignored by hardware. The subregister number of a source channel is provided by the subregister field of the corresponding address subregister.

Destination register may be either a directly-addressed or an indirectly-addressed register.

This instruction effectively performs a subfield shuffling from one register to another. Up to eight subfields can be selected by an instruction.

Restrictions:

Source operand must be a GRF register.

Source and destination must be the same type.

This instruction does not implicitly update accumulator register.

Execution size must be less than 16 (as there are only 8 address registers)

Address register for source must be aligned to the base (a0.0).

Destination register (directly or indirectly addressed) must be half GRF aligned (i.e. 16-byte aligned).

Destination stride in unit of byte must equal to the source element size in unit of byte.

Align16 access mode is not allowed for MOVI.

Alll the index registers used in MOVI instruction must all point to the same GRF register.

MOVI must use 1x1 indirect regioning.

MOVI is always based on register offset zero no matter what the destination offset is. The destination offset is used in HW only to create channel enables. The first index register (a0.0) will always be used to select the first element of the destination start from offset zero. Each index register will be used to select 1 element if type is byte, each index register will be used to select 2 elements if type is word, and each index register will be used to select 4 elements if type is dword

Conditional Modifier is not allowed for this instruction.

HW Implementation Details:

The destination offset of the *movi* instruction is only used in HW to generate destination write enables. Each element of the destination is directly mapped to the index registers for the *movi* instruction.

For Byte *movi*, byte0 of the destination is selected by (a0.0'[4:0]), byte1 is selected by (a0.1'[4:0]), ..., and byte7 is selected by (a0.7'[4:0]). The rest of the bytes are undefined.

For Word *movi*, byte0 of the destination is selected by (a0.0'[4:1] & "0"), byte1 is selected by (a0.0'[4:1] & "1"), byte2 is selected by (a0.1'[4:1] & "0"), byte3 is selected by (a0.1'[4:1] & "1"), ..., and byte15 is selected by (a0.7'[4:1] & "1"). The rest of the bytes are undefined.

For DWord or float *movi*, byte0 of the destination is selected by (a0.0'[4:2] & "00"), byte1 is selected by (a0.0'[4:2] & "01"), byte2 is selected by (a0.0'[4:2] & "10"), byte3 is selected by (a0.0'[4:2] & "11"), byte4 is selected by (a0.1'[4:2] & "00"), byte5 is selected by (a0.1'[4:2] & "01"), ..., byte31 is selected by (a0.7'[4:2] & "11").

For all 3 conditions above, a0.n'[4:0] = a0.n[4:0] + addr_imm[4:0].

8.3.31 mul – Multiply

Opcode	Instruction	Description
65 (0x41)	mul <dst> <src0> <src1>	Performing component-wise multiplication of <src0> and <src1> and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[FLT] [INT]	[FLT] [INT]

Format:

```
[(<pred>)] mul[.<cmod>] (<exec_size>) <dst> <src0> <src1>
```

Syntax:

```
[(<pred>)] mul[.<cmod>] (<exec_size>) reg reg reg
[(<pred>)] mul[.<cmod>] (<exec_size>) reg reg imm32
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = src0.chan[n] * src1.chan[n];
        }
}
```

Description:

The *mul* instruction performs component-wise multiplication of <src0> and <src1> and stores the results in <dst>.

[DevSNB]: When both <src0> and <src1> are of type D or UD, only the lower 16 bits of each element of <src0> are used. Accumulator maintains full 48-bit precision. The macro described in *mach* instruction should be used to obtain the full precision 64 bits multiplication results.

Multiplication of two floating point numbers follows rules in Table 8-5 (or

Table 8-6), if the current floating point mode is IEEE mode (or ALT mode).

Table 8-5. Floating point multiplication of A (column) and B (row) in IEEE mode

	–inf	**–finite**	**–1.0**	**–denorm**	**–0**	**+0**	**+denorm**	**+1.0**	**+finite**	**+inf**	**NaN**
–inf	+inf	+inf	+inf	NaN	NaN	NaN	NaN	–inf	–inf	–inf	NaN
–finite	+inf	*	–A	+0	+0	–0	–0	A	**	–inf	NaN
–1.0	+inf	–B	+1.0	+0	+0	–0	–0	–1.0	–B	–inf	NaN
–denorm	NaN	+0	+0	+0	+0	–0	–0	–0	–0	NaN	NaN
–0	NaN	+0	+0	+0	+0	–0	–0	–0	–0	NaN	NaN
+0	NaN	–0	–0	–0	–0	+0	+0	+0	+0	NaN	NaN
+denorm	NaN	–0	–0	–0	–0	+0	+0	+0	+0	NaN	NaN
+1.0	–inf	B	–1.0	–0	–0	+0	+0	+1.0	B	+inf	NaN
+finite	–inf	**	–A	–0	–0	+0	+0	A	*	+inf	NaN
+inf	–inf	–inf	–inf	NaN	NaN	NaN	NaN	+inf	+inf	+inf	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Note:

* Result may be {+finite, +inf (overflow)}

** Result may be {–inf (overflow), –finite}

Table 8-6. Floating point multiplication of A (column) and B (row) in ALT mode

	– fmax	**–finite**	**–1.0**	**–denorm**	**–0**	**+0**	**+denorm**	**+1.0**	**+finite**	**+fmax**	*****
– fmax	+fmax	+fmax	+fmax	**–0**	**–0**	**+0**	**+0**	–fmax	–fmax	–fmax	
–finite	+fmax	*	–A	+0	+0	–0	–0	A	**	–fmax	
–1.0	+fmax	–B	+1.0	+0	+0	–0	–0	–1.0	–B	–fmax	
–denorm	+0	+0	+0	+0	+0	–0	–0	–0	–0	–0	
–0	+0	+0	+0	+0	+0	–0	–0	–0	–0	–0	
+0	–0	–0	–0	–0	–0	+0	+0	+0	+0	+0	
+denorm	–0	–0	–0	–0	–0	+0	+0	+0	+0	+0	
+1.0	–fmax	B	–1.0	–0	–0	+0	+0	+1.0	B	+fmax	
+finite	–fmax	**	–A	–0	–0	+0	+0	A	*	+fmax	
+fmax	–fmax	–fmax	–fmax	–0	–0	+0	+0	+fmax	+fmax	+fmax	
*****											

Note:

* Result may be {+finite, +fmax (overflow)}

** Result may be {–fmax (overflow), –finite}

*** Result is undefined If any of A and/or is {–inf, +inf, NaN}

Restrictions:

Source operands cannot be an accumulator register.

When operating on integers with at least one of the source being a dword type (signed or unsigned), the destination cannot be a float (implementation note: the data converter only looks at the lower 34 bits of the results).

Dword integer source is not allowed for this instruction in float execution mode. In other words, if one source is of type float (:f, :vf), the other source cannot be of type dword integer (:ud or :d).

When operating on integers with at least one of the source being a dword type (signed or unsigned), the Overflow and Sign flags are undefined. Therefore, conditional modifier and instruction operation '.sat' cannot be used.

[DevSNB]: When multiple a DW and a W, the W has to be on src0, and the DW has to be on src1.

8.3.32 nop – No Operation

Opcode	Instruction	Description
126 (0x7E)	nop	Issuing an dummy instruction and performing no operation.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types

Format:

nop

Syntax:

nop

Pseudocode:

n/a

Description:

The *nop* instruction takes an instruction dispatch but performs no operation. It may be used for assembly patching in memory, or be used to insert an instruction delay in the program sequence.

The *nop* instruction takes no operands, no instruction modifier, no conditional modifier and no predication.

8.3.33 not – Logic Not

Opcode	Instruction	Description
4 (0x04)	not <dst> <src0>	Performing component-wise logic NOT of <src0> and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•		•	•	[INT]	[INT]

Format:

```
[(<pred>)] not[.<cmod>] (<exec_size>) <dst> <src0>
```

Syntax:

```
[(<pred>)] not[.<cmod>] (<exec_size>) reg reg
[(<pred>)] not[.<cmod>] (<exec_size>) reg imm32
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = !src0.chan[n]
        }
}
```

Description:

The *not* instruction performs logical NOT operation (or one's compliment) of <src0> and storing the results in <dst>.

Source modifiers are allowed.

Accumulator register is allowed to be the destination of this instruction with the restrictions listed below.

Restrictions:

This instruction does not work with float type operands.

Sign (SN) and Overflow (OF) conditions are undefined for this logic instruction. Consequently, saturation modifier (.sat) is not allowed.

This instruction does not implicitely update accumulator register.

Accumulator is allowed to be the source of this instruction, but source modifier is not allowed for an accumulator source.

When accumulator is the destination of this instruction, only the low bits corresponding to the data type (16 bits for word or 32 bits for dword integer instruction) in the accumulator contain the correct results. The internal extra-precision bits as well as the sign bit of the accumulator are undefined. Consequently, there are restrictions for subsequent instructions that use the data in the accumulator register created from the previous logical instruction.

- Only logical and data move instructions are allowed to source the accumulator. Results of other instructions (e.g. arithmetic or shift) are undefined.
- When the accumulator is the source of a data move (mov or sel) instruction, the destination operand must be of integer type (e.g. no conversion to float) and this instruction cannot have satuation instruction modifier.

8.3.34 or – Logic Or

Opcode	Instruction	Description
6 (0x06)	or <dst> <src0> <src1>	Performing component-wise logic OR of <src0> and <src1> and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•		•	•	[INT]	[INT]

Format:

```
[(<pred>)] or[.<cmod>] (<exec_size>) <dst> <src0> <src1>
```

Syntax:

```
[(<pred>)] or[.<cmod>] (<exec_size>) reg reg reg
[(<pred>)] or[.<cmod>] (<exec_size>) reg reg imm32
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = src0.chan[n] | src1.chan[n];
        }
}
```

Description:

The *or* instruction performs component-wise logic OR operation between <src0> and <src1> and stores the results in <dst>.

Source modifiers are allowed.

Accumulator register is allowed to be the destination of this instruction with the restrictions listed below.

Restrictions:

Sign (SN) and Overflow (OF) conditions are undefined for this logic instruction. Consequently, saturation modifier (.sat) is not allowed.

This instruction does not work with float type operands.

Accumulator is allowed to be the source of this instruction, but source modifier is not allowed for an accumulator source.

This instruction does not implicitely update accumulator register.

When accumulator is the destination of this instruction, only the low bits corresponding to the data type (16 bits for word or 32 bits for dword integer instruction) in the accumulator contain the correct results. The internal extra-precision bits as well as the sign bit of the accumulator are undefined. Consequently, there are restrictions for subsequent instructions that use the data in the accumulator register created from the previous logical instruction.

- Only logical and data move instructions are allowed to source the accumulator. Results of other instructions (e.g. arithmetic or shift) are undefined.
- When the accumulator is the source of a data move (mov or sel) instruction, the destination operand must be of integer type (e.g. no conversion to float) and this instruction cannot have satuation instruction modifier.

8.3.35 pln – Plane

Opcode	Instruction	Description
90 (0x5A)	PLN <dst> <src0> <src1>	Computing a component-wise plane equation (w = p*u+q*v+r) of scalar (p, q, r) from <src0> and vector <src1> (and implied vector <src2>) and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[FLT]	[FLT]

Format:

```
[(<pred>)] pln[.<cmod>] (<exec_size>) <dst> <src0> <src1>
```

Syntax:

```
[(<pred>)] pln[.<cmod>] (<exec_size>) reg reg reg
```

Pseudocode:

```
Evaluate(WrEn);
```

```
for (n = 0; n < exec_size; n++) {
for (n = 0; n < exec_size; n++) {
        float dwP = src0.RegNum.SubRegNum[bits4:2] // a dword aligned scalar
        float dwQ = src0.RegNum.(SubRegNum[bit4:2]|0x1)   // 2nd component
        float dwR = src0.RegNum.(SubRegNum[bit4:2]|0x3)   // 4th component
        src2 = src1.(RegNum + 1)                          // Next GRF register
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = dwP * src1.chan[n] + dwQ * src2.chan[n] + dwR
        }
}
```

Description:

The *pln* instruction computes a component-wise plane equation (*w* = *p*u+q*v+r* where *u/v/w* are vectors and *p/q/r* are scalars) of <src0> and <src1> and storing the results in <dst>. <src1> is the input vector *u*. The second input vector v is implied from <src1>, as the next adjacent GRF register. <src0> provides input scalars *p, q* and *r*, where *p* is the scalar value based on the region description of <src0> and *q* and *r* are the scalar values implied from <src0> region. Specifically, *q/r* is the second/fourth component of the 4-tuple (128-bit aligned) that *p* belongs to.

When *pln* instruction is used in SIMD16 form, the same input data channels *p/q/r* for <src0> are used for both SIMD8 instructions. However, as <src1> has two vectors *u/v*, where *v* is implied, the second SIMD8 instruction takes src1.(RegNum+2) as the second source operand.

Restrictions:

This is a specialized instruction that only support execution size of 8 or 16.

<src0> region must be a replicated scalar (with HorzStride = VertStride = 0).

<src0>, <src1>, <src2>, and <dst> may be float.

Source operands cannot be an accumulator register.

<src0> for *pln* instruction has to has .0 or .4 as the subregister number.

[DevSNB]:<src1> must be even register aligned.

8.3.36 ret – Return [DevSNB]

Opcode	Instruction	Description
45 (0x2d)	ret NULL <src0>	Return back from a subroutine.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•				DWord	

Format:

[(<pred>)] ret (<exec_size>) NULL <src0>

Syntax:

[(<pred>)] ret (<exec_size>) NULL reg

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < 16; n++) {
        if (WrEn.channel[n] == 1) {
                PcIP[n] = src0.chan[0];
                CallMask[n] = 0;
        } else {
                PcIP[n] = IP + 1;
        }
}

if (CallMask[n:0] == 0)) { // all channels are zero
        Jump(src0.chan[0]);
        CallMask = src0.chan[1];
}
```

Description:

The *ret* instruction returns the execution from a subroutine back.

The *ret* instruction can be predicated or non-predicated. If non-predicated, all the channels will jump to the return IP in the first channel of src0. The CallMask will be restored from the second channel of src0. If predicated, the enabled channels will jump to the return IP in the first channels of src0, the corresponding

bit in the CallMask will be reset to zero, if all the bits in the CallMask are zero after the *call* instruction, the execution will jump to the return IP in the first channel of src0.

If SPF is ON, none of the PcIP is updated.

When SPF is on, the predication control must be scalar.

Restrictions:

This instruction cannot take accumulator as source.

8.3.37 rndd – Round Down

Opcode	Instruction	Description
69 (0x45)	rndd <dst> <src0>	Taking component-wise floating point downward rounding of <src0> and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[FLT]	[FLT]

Format:

[(<pred>)] rndd[.<cmod>] (<exec_size>) <dst> <src0>

Syntax:

[(<pred>)] rndd[.<cmod>] (<exec_size>) reg reg
[(<pred>)] rndd[.<cmod>] (<exec_size>) reg imm32

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = floor(src0.chan[n]);
        }
}
```

Description:

The *rndd* instruction takes component-wise floating point downward rounding (to the integral float number closer to negative infinity) of <src0> and storing the rounded integral float results in <dst>. This is commonly referred to as the *floor()* function.

This instruction only applies to floating point source and destination operands.

Output data <dst> for floating point rounding-down follow rules in Table 8-7 (or Table 8-8), if the current floating point mode is IEEE mode (or ALT mode).

Table 8-7. Floating point round-down in IEEE mode

<src0>	**–inf**	**–finite**	**–denorm**	**–0**	**+0**	**+denorm**	**+finite**	**+inf**	**NaN**
<dst>	–inf	*	–0	–0	+0	+0	**	+inf	NaN

Notes:

* Result may be {–finite, –0}.

** Result may be {+finite, +0}.

Table 8-8. Floating point round-down in ALT mode

<src0>	**– fmax**	**–finite**	**–denorm**	**–0**	**+0**	**+denorm**	**+finite**	**+ fmax**	*****
<dst>	–fmax	*	–0	–0	+0	+0	**	+fmax	

Notes:

* Result may be {–finite, –0}.

** Result may be {+finite, +0}.

*** Result is undefined if <src0> is {–inf, +inf, NaN}.

Restrictions:

This instruction cannot take accumulator as source or destination operand. However, when the accumulator is implicitly updated by this instruction, the results in the accumulator are undefined.

8.3.38 rndu – Round Up

Opcode	Instruction	Description
68 (0x44)	rndu <dst> <src0>	Taking component-wise floating point upward rounding of <src0> and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[FLT]	[FLT]

Format:

```
[(<pred>)] rndu[.<cmod>] (<exec_size>) <dst> <src0>
```

Syntax:

```
[(<pred>)] rndu[.<cmod>] (<exec_size>) reg reg
[(<pred>)] rndu[.<cmod>] (<exec_size>) reg imm32
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                if (src0.chan[n]-floor(src0.chan[n]) > 0.0f)
                        dst.chan[n] = floor(src0.chan[n]) + 1;
                else
                        dst.chan[n] = src0.chan[n];
        }
}
```

Description:

The *rndu* instruction takes component-wise floating point upward rounding (to the integral float number closer to positive infinity) of <src0>, commonly known as the *ceiling()* function.

This instruction only applies to floating point source and destination operands.

Table 8-9. Floating point round-up in IEEE mode

<src0>	**–inf**	**–finite**	**–denorm**	**–0**	**+0**	**+denorm**	**+finite**	**+inf**	**NaN**
<dst>	–inf	*	–0	–0	+0	+0	**	+inf	NaN

Notes:

* Result may be {–finite, –0}.

** Result may be {+finite, +0}.

*** Increment may be {0, 1}. It is 0 if source data is an integral float, and is 1 otherwise.

Table 8-10. Floating point round-up in ALT mode

<src0>	**– fmax**	**–finite**	**–denorm**	**–0**	**+0**	**+denorm**	**+finite**	**+ fmax**	********
<dst>	–fmax	*	–0	–0	+0	+0	**	+fmax	

Notes:

* Result may be {–finite, –0}.

** Result may be {+finite, +0}.

*** Increment may be {0, 1}. It is 0 if source data is an integral float, and is 1 otherwise.

**** Result is undefined if <src0> is {–inf, +inf, NaN}.

8.3.39 rnde – Round to Even

Opcode	Instruction	Description
70 (0x46)	Rnde <dst> <src0>	Taking component-wise floating point round-to-even operations of <src0> and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[FLT]	[FLT]

Format:

```
[(<pred>)] rnde[.<cmod>] (<exec_size>) <dst> <src0>
```

Syntax:

```
[(<pred>)] rnde[.<cmod>] (<exec_size>) reg reg
[(<pred>)] rnde[.<cmod>] (<exec_size>) reg imm32
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = floor(src0.chan[n]);
                if (src0.chan[n]-floor(src0.chan[n]) > 0.5f) {
                        dst.chan[n] = floor(src0.chan[n]) + 1;
                } else if (src0.chan[n]-floor(src0.chan[n]) < 0.5f) {
                        dst.chan[n] = floor(src0.chan[n]);
                } else {
                        if (dst.chan[n] is odd) {
                                dst.chan[n] = floor(src0.chan[n]) + 1;
                        } else {
                                dst.chan[n] = floor(src0.chan[n]);
                        }
                }
        }
}
```

Description:

The *rnde* instruction takes component-wise floating point round-to-even operation of <src0> with results in two pieces – a downward rounded integral float results stored in <dst> and the round-to-even increments

stored in the rounding increment bits. The round-to-even increment must be added to the results in <dst> to create the final round-to-even values to emulate the round-to-even operation, commonly known as the *round()* function. The final results are the one of the two integral float values that is nearer to the input values. If the neither possibility is nearer, the even alternative is chosen.

Output data <dst> for floating point rounding-to-even follow rules in Table 8-11 (or Table 8-12), if the current floating point mode is IEEE mode (or ALT mode).

Table 8-11. Floating point round-to-even in IEEE mode

<src0>	**–inf**	**–finite**	**–denorm**	**–0**	**+0**	**+denorm**	**+finite**	**+inf**	**NaN**
<dst>	–inf	*	–0	–0	+0	+0	**	+inf	NaN

Notes:

* Result may be {–finite, –0}.

** Result may be {+finite, +0}.

*** Increment may be {0, 1}. It is 0 if source data is an integral float. It may be 0 or 1 otherwise.

Table 8-12. Floating point round-to-even in ALT mode

<src0>	**– fmax**	**–finite**	**–denorm**	**–0**	**+0**	**+denorm**	**+finite**	**+ fmax**	********
<dst>	–fmax	*	–0	–0	+0	+0	**	+fmax	

Notes:

* Result may be {–finite, –0}.

** Result may be {+finite, +0}.

*** Increment may be {0, 1}. It is 0 if source data is an integral float. It may be 0 or 1 otherwise.

**** Result is undefined if <src0> is {–inf, +inf, NaN}.

8.3.40 rndz – Round to Zero

Opcode	Instruction	Description
71 (0x47)	rndz <dst> <src0>	Taking component-wise floating point round-to-zero operations of <src0> and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[FLT]	[FLT]

Format:

```
[(<pred>)] rndz[.<cmod>] (<exec_size>) <dst> <src0>
```

Syntax:

```
[(<pred>)] rndz[.<cmod>] (<exec_size>) reg reg
[(<pred>)] rndz[.<cmod>] (<exec_size>) reg imm32
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = floor(src0.chan[n]);
                if (abs(src0.chan[n]) < abs(dst.chan[n])) {
                        dst.chan[n] = floor(src0.chan[n]) + 1;
                } else {
                        dst.chan[n] = floor(src0.chan[n]);
                }
        }
}
```

Description:

The *rndz* instruction takes component-wise floating point round-to-zero operation of <src0> with results in two pieces – a downward rounded integral float results stored in <dst> and the round-to-zero increments stored in the rounding increment bits. The round-to-zero increment must be added to the results in <dst> to create the final round-to-zero values to emulate the round-to-zero operation, commonly known as the *truncate()* function. The final results are the one of the two closest integral float values to the input values that is nearer to zero.

Output data <dst> for floating point rounding-to-zero follow rules in Table 8-13 (or Table 8-14), if the current floating point mode is IEEE mode (or ALT mode).

Table 8-13. Floating point round-to-zero in IEEE mode

<src0>	**–inf**	**–finite**	**–denorm**	**–0**	**+0**	**+denorm**	**+finite**	**+inf**	**NaN**
<dst>	–inf	*	–0	–0	+0	+0	**	+inf	NaN

Notes:

* Result may be {–finite, –0}.

** Result may be {+finite, +0}.

*** Increment may be {0, 1}. It is 0 if source data is an integral float, and is 1 otherwise.

Table 8-14. Floating point round-to-zero in ALT mode

<src0>	**– fmax**	**–finite**	**–denorm**	**–0**	**+0**	**+denorm**	**+finite**	**+ fmax**	********
<dst>	–fmax	*	–0	–0	+0	+0	**	+fmax	

Notes:

* Result may be {–finite, –0}.

** Result may be {+finite, +0}.

*** Increment may be {0, 1}. It is 0 if source data is an integral float, and is 1 otherwise.

**** Result is undefined if <src0> is {–inf, +inf, NaN}.

8.3.41 sad2 – Sum of Absolute Difference 2

Opcode	Instruction	Description
80 (0x50)	sad2 <dst> <src0> <src1>	Performing a two-wide sum-of-absolute-difference operation on a 2-tuple basis of <src0> and <src1>, and storing the scalar result to the first channel per 2-tuple in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[INT]	[INT]

Format:

```
[(<pred>)] sad2[.<cmod>] (<exec_size>) <dst> <src0> <src1>
```

Syntax:

```
[(<pred>)] sad2[.<cmod>] (<exec_size>) reg reg reg
[(<pred>)] sad2[.<cmod>] (<exec_size>) reg reg imm32
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n+=2) {
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = abs(src0.chan[n] - src1.chan[n])
                        + abs(src0.chan[n+1] - src1.chan[n+1]);
}
```

Description:

The *sad2* instruction takes source data channels from <src0> and <src1> in groups of 2-tuples. For each 2-tuple, it computes the sum-of-absolute-difference (SAD) between <src0> and <src1> and stores the scalar result in the first channel of the 2-tuple in <dst>.

This instruction only applies to integer operands. In particular, source operands must be unsigned bytes and/or signed bytes and destination operand must be of word type. Source modifiers are allowed.

The results are also stored in the accumulator register. Destination operand and accumulator maintain 16-bit per channel precision.

Destination register must have a stride of 2 words and must be aligned to even word. The even words in destination region will contain the correct data. The odd words are also written but with undefined values

Restrictions:

Source operands cannot be an accumulator register.

Execution size cannot be 1 as the computation requires at least two data channels.

8.3.42 sada2 – Sum of Absolute Difference Accumulate 2

Opcode	Instruction	Description
81 (0x51)	sada2 <dst> <src0> <src1>	Performing a two-wide sum-of-absolute-difference operation on a 2-tuple basis of <src0> and <src1>, added to that from the accumulator, and storing the scalar result to the first channel per 2-tuple in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[INT]	[INT]

Format:

```
[(<pred>)] sada2[.<cmod>] (<exec_size>) <dst> <src0> <src1>
```

Syntax:

```
[(<pred>)] sada2[.<cmod>] (<exec_size>) reg reg reg
[(<pred>)] sada2[.<cmod>] (<exec_size>) reg reg imm32
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n+=2) {
        uwTmp = abs(src0.channel[n] - src1.channel[n])
                + abs(src0.channel[n+1] - src1.channel[n+1])
        if (WrEn.channel[n] == 1) {
                dst.channel[n] = uwTmp + acc[n]
        }
}
```

Description:

The *sada2* instruction takes source data channels from <src0> and <src1> in groups of 2-tuples. For each 2-tuple, it computes the sum-of-absolute-difference (SAD) between <src0> and <src1>, adds the intermediate result with the accumulator value corresponding to the first channel, and stores the scalar result in the first channel of the 2-tuple in <dst>.

This instruction only applies to integer operands. In particular, source operands must be unsigned bytes and/or signed bytes and destination operand must be of word type. Source modifiers are allowed.

The results are also stored in the accumulator register. Destination operand and accumulator maintain 16-bit per channel precision. Higher precision (guide bits) stored in the accumulator allows multiple rounds (64 rounds) of sada2 instructions to be issued back to back without overflow the accumulator.

Destination register must have a stride of 2 words and must be aligned to even word. The even words in destination region will contain the correct data. The odd words are also written but with undefined values

Restrictions:

Source operands cannot be an accumulator register.

Execution size cannot be 1 as the computation requires at least two data channels.

8.3.43 sel – Select

Opcode	Instruction	Description
2 (0x02)	(pred) sel <dst> <src0> <src1>	Component-wise selective move from <src0> or <src1> to <dst> based on predication or cmod result. The sel instruction can not use accumulator as destination

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[FLT] [INT]	[FLT] [INT]

Format:

(<pred>) sel[.<cmod>] (<exec_size>) <dst> <src0> <src1>

Syntax:

(<pred>) sel[.<cmod>] (<exec_size>) reg reg reg
(<pred>) sel[.<cmod>] (<exec_size>) reg reg imm32

Pseudocode:

```
Evaluate(WrEn, NoPMask);

if (cmod == "0000") { // no CMod
        Evaluate(PMask);
        for (n = 0; n < exec_size; n++) {
                if (WrEn.channel[n] == 1) {
                        if (PMask.channel[n] == 1) {
                                dst.channel[n] = src0.channel[n]
                        } else {
                                dst.channel[n] = src1.channel[n]
                        }
                }
        }
```

```
}
else { // with CMod
        Evaluate(CMod);
        for (n = 0; n < exec_size; n++) {
                if (WrEn.channel[n] == 1) {
                        if (CMod.channel[n] == 1) {
                                dst.channel[n] = src0.channel[n]
                        } else {
                                dst.channel[n] = src1.channel[n]
                        }
                }
        }
}
```

Description:

The *sel* instruction selectively moves the components in <src0> or <src1> into the channels of <dst> based on the predication. On a channel by channel basis, if the channel condition is true, data in <src0> is moved into <dst>; Otherwise, data in <src1> is moved into <dst>.

As the predication is used to select the two sources, it is not included in the evaluation of WrEn. <pred> is mandatory if <cmod> is “0000”. If it is <omitted> and <cmod> is “0000”, the results are unpredictable.

In case of <cmod> not equal to “0000”, a compare will be performed and the result flag will be used for the *sel* instruction. <cmod> .ge and .l follow the CMPN rules, and all other <cmod> follow the CMP rules. <pred> is not allowed in this mode.

sel instruction with <cmod> .l should be used to emulate MIN instruction.

sel instruction with <cmod> .ge should be used to emulate MAX instruciton.

If any of the source of the *sel* instruction is NaN, the non-NaN source will be the result, in case both sources are NaN, the result will also be NaN. This only applies to *sel* instruction with .l and .ge conditional modifier. For the other conditional modifiers, src1 will be always be selected if either or both sources is/are NaN.

sel instruction with <cmod> will flush denorm to zero; *sel* instruction without <cmod> will retain denorm.

Format conversion is not allowed.

[DevSNB]: sel instruciton can not use accumulator source.

Restrictions:

Destination channels cannot be on odd-byte sub-register locations. In other words, when destination is of byte type, destination horizontal stride cannot be 1. If destination horizontal stride is not 1, destination register region origin cannot be on an odd byte location. This is because that the conditional flag for execution channels that have minimal granularity of word are used by this instruction.

[DevSNB]: The sel instruction can not use accumulator as destination

8.3.44 send – Send Message [DevSNB]

Opcode	Instruction	Description
49 (0x31)	send <dest> <src> <ex_desc> <desc>	Send a message stored in MRF starting at <src> to a shared function identified by <ex_desc> along with control from <desc> with a GRF writeback location at <dest>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•					[FLT] [INT]

Format:

[(<pred>)] send (<exec_size>) <dest> <src> <ex_desc> <desc>

Syntax:

[(<pred>)] send (<exec_size>) reg mreg imm6 reg32a

[(<pred>)] send (<exec_size>) reg mreg imm6 imm32

Pseudocode:

```
Evaluate(WrEn);
<MsgChEnable> = WrEn;
<SourceReg> = <src>.RegNum;
MessageEnqueue(<MsgChEnable>, <ResponseReg>, <SourceReg>, <desc>,
<ex_dest>);
```

Description:

The *send* instruction performs data communication between a thread and external function units, including shared functions (Sampler, Data Port Read, Data Port Write, URB, and Message Gateway) and some fixed functions (e.g. Thread Spawner, who also have an unique Shared Function ID). The *send* instruction adds an entry to the EU's message request queue. The request message is stored in a block of contiguous MRF registers. The response message, if present, will be returned to a block of contiguous GRF registers. The return GRF writes may be in any order depending on the external function units. <src> is the lead MRF register for request. <dest> is the lead GRF register for response. The message descriptor field <desc> contains the Message Length (the number of consecutive MRF registers) and the Response Length (the number of consecutive GRF registers). It also contains the header present bit, and the function control signals. The extend mesage descriptor field <ex_desc> contains the target function ID. WrEn is forwarded to the target function in the message sideband.

The *send* instruction is the only way to terminate a thread. When the EOT (End of Thread) bit of <desc> is set, it indicates the end of thread to the EU, the Thread Dispatcher and, in most cases, the parent fixed function.

Message descriptor field <desc> can be a 32-bit immediate, *imm32*, or a 32-bit scalar register, <reg32a>. DevSNB restricts that the 32-bit scalar register <reg32a> must be the leading dword of the address register. It should be in the form of a0.0<0;1,0>:ud. When <desc> is a register operand, only the lower 29 bits of <reg32a> are used.

<ex_desc> is a 6-bit immediate, imm6. The lower 4bits of the <ex_desc> specifies the SFID for the message. The MSb of the message descriptor, the EOT field, always comes from bit 127 of the instruction word, which is the MSb of imm6. A thread must terminate with a *send* instruction with EOT turned on.

<src> is a 256-bit aligned MRF register. It serves as the leading MRF register of the request.

<dest> serves for two purposes: to provide the leading GRF register location for the response message if present, and to provide parameters to form the channel enable sideband signals.

<dest> signals whether there is a response to the message request. It can be either a null register, a direct-addressed GRF register or a register-indirect GRF register. Otherwise, hardware behavior is undefined.

If <dest> is null, there is no response to the request. Meanwhile, the Response Length field in <desc> must be 0. Certain types of message requests, such as memory write (store) through the Data Port, do not want response data from the function unit. If so, the posted destination operand can be null.

If <dest> is a GRF register, the register number is forwarded to the shared function. In this case, the target function unit must send one or more response message phases back to the requesting thread. The number of response message phases must match the Response Length field in <desc>, which of course cannot be zero. For some cases, it could be an empty return message. An empty return message is defined as a single phase message with all channel enables turned off.

The subregister number, horizontal stride, destination mask and type fields of <dest> are always valid and are used in part to generate the WrEn. This is true even if <dest> is a null register (this is an exception for null as for most cases these fields are ignored by hardware). These parameters of <dest> follow the same restriction as that of normal destination operand – destination region cannot cross the 256-bit register boundary.

The 16-bit channel enables of the message sideband are formed based the WrEn. Interpretation of the channel enable sideband signals is subject to the target external function. In general for a ‘send’ instruction with return messages, they are used as the destination dword write mask for the GRF registers starting at <dest>. For a message that has multiple return phases, the same set of channel enable signals applies to all the return phases.

Thread managed memory coherency: A special usage of using non-null <dest> is to support write-commit signaling for memory write service by the Data Port Write unit. If <post_dest> is not null for a memory write request, the Data Port along with the Data Cache or Render Cache will wait until all the posted writes for the request have reached the coherent domain before sending back to the requesting thread an empty message to <dest> register. A memory write reaching the coherent domain, also referred to as reaching the global observable state, means that subsequent read to the same memory location, no matter which thread issues the read, must return the data of the write.

The destination dependency control, {NoDDClr}, can be used in this instruction. This allows software to control the destination dependencies for multiple ‘read’-type messages similar to that for multiple instructions using EU execution pipeline. As *send* does not check register dependencies for the post destination, {NoDDChk} should not be used for this instruction.

DevSNB supports both header and headerless messages, and it's indicated by the "Header Present" bit in the message descriptor.

Table 8-15. Message Descriptor Definition

Bit	Description
31:29	Reserved : MBZ
28:25	**Message Length.** This field specifies the number of 256-bit MRF registers starting from <curr_dest> to be sent out on the request message payload. Valid value ranges from 1 to 15. A value of 0 is considered erroneous. Format = U4 Range = [1,15]
24:20	**Response Length.** This field indicates the number of 256-bit registers expected in the message response. The valid value ranges from 0 to 16. A value 0 indicates that the request message does not expect any response. The largest response supported is 15 GRF registers. Format = U5 Range = [0,16]
19	**Header Present.** If set, indicates that the message includes a header. Depending on the target shared function, this field may be restricted to either enabled or disabled. Refer to the specific shared function section for details. Format = Enable
18:0	**Function Control** This field is intended to control the target function unit. Refer to the section on the specific target function unit for details on the contents of this field.

Table 8-16. Extended Message Descriptor Definition

Bit	Description
5	**End Of Thread** This field, if set, indicates that this is the final message of the thread and the thread's resources can be reclaimed.
4	reserved
3:0	**Target Function ID** This field indicates the function unit for which the message is intended. 0000: Null 0001: Reserved

Bit	Description
	0010: Sampling Engine 0011: Message Gateway 0100: Data Port Sampler Cache 0101: Data Port Render Cache 0110: URB 0111: Thread Spawner 1000: Video Motion Estimation 1001: Constant Cache 1010 – 1111: Reserved

The following table provides a summary of the signals associated with each message that is sent to the shared function. It contains fields from the send instruction as well as fields from control register *cr0* and state register *sr0*.

Table 8-17. Sideband Signals Associated with Each Message Sent to the Shared Function

Signal	Bits	Source
EOT	1	End of Thread: Sourced from the EOT bit in ***send*** instruction word
SFID	3	Shared Function Identifier: Sourced from the target function ID field in <ex_desc> of ***send***
MLEN	4	Message Length: Sourced from the message length field in <desc> of ***send***
RLEN	5	Response Length: Sourced from the response length field in <desc> of ***send***
FC	19	Function Control: Sourced from the function control field in <desc> of ***send***
REG	7	Destination Register: Sourced from the 256-bit register aligned register number of the <dest> field of ***send***
CE	16	Channel Enable: Sourced from the write enable of ***send***
CLEAR	1	Destination Register Clear: Source from the Destination Dependency Control field (inverse of NoDDClr) in ***send*** instruction word
FFID	4	Fixed Function Identifier: Sourced from the Fixed Function ID field in *sr0*
EUID	4	Execution Unit Identifier: Sourced from the EUID field in *sr0*
TID	2	Thread Identifier: Sourced from the TID field in *sr0*

Restrictions:

Software must obey the following rules in signaling the end of thread using the *send* instruction:

- The posted destination operand must be null.
    - No acknowledgement is allowed for the *send* instruction that signifies the end of thread. This is to avoid deadlock as the EU is expecting to free up the terminated thread's resource.
- A thread must terminate with a *send* instruction with message to a shared function on the output message bus; therefore, it cannot terminate with a *send* instruction with message to the following shared functions: Sampler unit, NULL function
- For example, a thread may terminate with a URB write message or a render cache write message.
- A root thread originated from the media (generic) pipeline must terminate with a *send* instruction with message to the Thread Spawner unit. A child thread should also terminate with a send to TS. Please refer to the Media Chapter for more detailed description.

The *send* instruction can not update accumulator registers.

Saturate is not supported for *send* instruction.

ThreadCtrl are not supported for *send* instruction.

R0 or NULL register can not be used as destination for all messages with writebacks

[Errata: DevSNB(SNB)] If MRF register is updated by any instruction that "indexed/indirect" source AND is followed by a *send,* the instruction requires a "Switch". This is to avoid race condition where *send* may dispatch before MRF is updated.

8.3.45 sendc – Conditional Send Message

Opcode	Instruction	Description
49 (0x31)	sendc <dest> <src> <desc> <ex_dest>	Wait for the dependencies in the TDR register cleared, then send a message stored in MRF starting at <str> to a shared function identified by <ex_desc> along with control from <desc> with a GRF writeback location at <dest>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•					[FLT] [INT]

Format:

[(<pred>)] sendc (<exec_size>) <dest> <src> <exdesc> <desc>

Syntax:

[(<pred>)] sendc (<exec_size>) reg reg reg32a imm4
[(<pred>)] sendc (<exec_size>) reg reg imm32 imm4

Pseudocode:

```
if ((TDR[7]... || TDR[2] || TDR[1] || TDR[0]) == TRUE) {
        wait;
}
Evaluate(WrEn);
<MsgChEnable> = WrEn;
<SourceReg> = <src>.RegNum;
MessageEnqueue(<MsgChEnable>, <ResponseReg>, <SourceReg>, <desc>,
<ex_dest>);
```

Description:

The *send*c instruction has the same behavior as the *send* instruction except the following.

The *sendc* instruction first check the dependent threads inside the Thread Dependency Register, there are up to 4 dependent threads in the TDR register. The *sendc* instruction will be executed only when all the dependent threads in the TDR register are retired.

Restrictions:

The *sendc* instruction has the same restrictions as the *send* instruction.

8.3.46 shl – Shift Left

Opcode	Instruction	Description
9 (0x09)	shl <dst> <src0> <src1>	Performing component-wise logic left shift of <src0> and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[INT]	[INT]

Format:

```
[(<pred>)] shl[.<cmod>] (<exec_size>) <dst> <src0> <src1>
```

Syntax:

```
[(<pred>)] shl[.<cmod>] (<exec_size>) reg reg reg
[(<pred>)] shl[.<cmod>] (<exec_size>) reg reg imm32
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = src0.chan[n] << src1.chan[n]
        }
}
```

Description:

The *shl* instruction performs component-wise logical left shift of <src0> with zero-insertion and storing the results in <dst>. The amount of bit shift is provided by <src1>, where only the 5 LSBs of each channel of <src1> are used as an unsigned integer value. The MSBs of <src1> data channels are ignored. . The destination operand cannot be the accumulator.

5-bit shifting applies to packed-dword mode and packed-word mode. For packed word mode, the accumulators have 34 bits per channel. <src0> and <dst> can be signed or unsigned integers and can be of different types. This instruction does not work with float type operands. Saturation modifier is only allowed when this instruction is in packed-word mode. Hardware detects overflow properly and use it to perform saturation operation on the output, as long as the shifted result is within 34 bits. Otherwise, the result is undefined.

Results of saturation in packed-dword mode are unpredicable.

Restrictions:

This instruction does not work with float type operands.

8.3.47 shr – Shift Right

Opcode	Instruction	Description
8 (0x08)	shr <dst> <src0> <src1>	Performing component-wise logic right shift of <src0> and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•	•	•	•	[INT]	[INT]

Format:

```
[(<pred>)] shr[.<cmod>] (<exec_size>) <dst> <src0> <src1>
```

Syntax:

```
[(<pred>)] shr[.<cmod>] (<exec_size>) reg reg reg
[(<pred>)] shr[.<cmod>] (<exec_size>) reg reg imm32
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = src0.chan[n] >> src1.chan[n]
        }
}
```

Description:

The *shr* instruction performs component-wise logical right shift of <src0> with zero-insertion and storing the results in <dst>. The amount of bit shift is provided by <src1> where only the 5 LSBs of each channel of <src1> are used as an unsigned integer value. The MSBs of <src1> data channels are ignored.

5-bit shifting applies to packed-dword mode and packed-word mode. For packed word mode, the accumulators have 33 bits per channel.

This instruction only takes on unsigned sources. When <src0> contains unsigned integers, no source modifier is allowed. <src0> is only allowed to be signed integer if source modifier (abs) is used. *Note: for unsigned sources, the behavior of shr and asr are effectively the same.*

Restrictions:

This instruction does not work with float type operands.

8.3.48 wait – Wait Notification

Opcode	Instruction	Description
48 (0x30)	wait <nreg>	Waiting for notification on the notification register <nreg>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types

Format:

wait (<exec_size>) <nreg>

Syntax:

wait (1) n#

Pseudocode:

n/a

Description:

The *wait* instruction evaluates the value of the notification count register <nreg>. If <nreg> is zero, the execution of the thread is stalled and the thread is put in 'wait_for_notification' state. If <nreg> is not zero (i.e., one or more notifications have been received), <nreg> is decremented by one and the thread continues executing on the next instruction. If a thread is in the 'wait_for_notification' state, when a notification arrives, the notification count register is incremented by one. As the notification count register becomes non-zero, the thread wakes up to continue execution and at the same time the notification register is decremented by one. If there was only one notification arrived, the notification register value becomes zero. However, during the above mentioned time period, it is possible that more notifications may arrive, making the notification register non-zero again.

When multiple notifications are received, software must use 'wait' instruction to decrement notification count register for each of the notifications.

Notification register n0:ud is for thread to thread communication (through message gateway shared function) and n1:ud for host to thread communication (through MMIO registers). See Message Gateway chapter for thread-thread communication.

Restrictions:

Only one source operand.

<src0> and <dst> must be n0, n1, or n2, <src1> must be *null*.

Execution size must be 1 as the notification registers are scalar.

Predication is not allowed.

Implementation restriction:

Two back-to-back *wait* instructions in a program (without any instruction in between) are not allowed. As a minimal, a *nop* has to be inserted between two *wait* instructions.

8.3.49 while – While [DevSNB]

Opcode	Instruction	Description
39 (0x27)	while <dst> <src0> <src1> <JIP>	Marking the end of a do-while block of code.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•					

Format:

```
[(<pred>)] while (<exec_size>) null null null <JIP>
        while (<exec_size>) null <src0> <src1> <JIP>
```

Syntax:

```
[(<pred>)] while (<exec_size>) null null null imm16
        while (<exec_size>) null reg reg imm16
        while (<exec_size>) null reg imm32 imm16
```

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < 16; n++) {
        if (WrEn.channel[n] == 1) {
                if (<cmod> == 0) { // no embedded compare
                        if (PMask.channel[n] == 1) {
                                PcIP[n] = IP + <JIP>;
                        }
                        else {
                                PcIP[n] = IP + 1;
                        }
                }
                else { // with embedded compare
                        if (cmod.channel[n] == 1) {
                                PcIP[n] = IP + <JIP>;
                        }
                        else {
```

```
                                        PcIP[n] = IP + 1;
                                }
                        }
                }
        }
        if (<cmod> == 0) { // no embedded compare
                if (|PMask == 1) { // any enabled channel true
                        Jump(IP + <JIP>);
                }
        }
        else { // with embedded compare
                if (|cmod == 1) { // any enabled channel true
                        Jump(IP + <JIP>);
                }
        }
```

Description:

The *while* instruction marks the end of a do-while block. The instruction first evaluates the loop termination condition for each channel based the current channel enables the predication flag specified in the instruction. If any channel has not terminated, a branch is taken to a destination address based specified in the instruction, and the loop continued for those channels. Otherwise, execution continue down to the next instruction.

The following table describes the 16-bit jump target offset <JIP>. <JIP> is a signed 16-bit number, added to IP pre-increment, and should point to the first instruction with the *do* label of the do-while block of code. It should be a negative number for the backward referencing. In DevSNB binary, <JIP> is at location <dst> and must be of type W (signed word integer).

Bit	Description
15:0	**JIP (Jump Target Offset).** This field specifies the jump distance in number of 64bits data chunks if a jump is taken for the instruction. Format = S15. Signed integer in 2's compliment

If SPF is ON, none of the PcIP is updated.

Restrictions:

To use embedded compares, the predicate control field for this instruction must be zero, and the conditional modifier field must be none zero. The destination must follow the rules below:

5. Must has the same element size as source0
6. Must have horizontal stride of 1

To use predicated *if* instruction, the conditional modifier field must be zero.

8.3.50 xor – Logic Xor

Opcode	Instruction	Description
7 (0x07)	xor <dst> <src0> <src1>	Performing component-wise logic XOR of <src0> and <src1> and storing the results in <dst>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Types
•		•	•	[INT]	[INT]

Format:

[(<pred>)] xor[.<cmod>] (<exec_size>) <dst> <src0> <src1>

Syntax:

[(<pred>)] xor[.<cmod>] (<exec_size>) reg reg reg
[(<pred>)] xor[.<cmod>] (<exec_size>) reg reg imm32

Pseudocode:

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
        if (WrEn.chan[n] == 1) {
                dst.chan[n] = src0.chan[n] ^ src1.chan[n];
        }
}
```

Description:

The *xor* instruction performs component-wise logic XOR operation between <src0> and <src1> and stores the results in <dst>.

Source modifiers are allowed.

Accumulator register is allowed to be the destination of this instruction with the restrictions listed below.

Restrictions:

Sign (SN) and Overflow (OF) conditions are undefined for this logic instruction. Consequently, saturation modifier (.sat) is not allowed.

This instruction does not work with float type operands.

Accumulator is allowed to be the source of this instruction, but source modifier is not allowed for an accumulator source.

When accumulator is the destination of this instruction, only the low bits corresponding to the data type (16 bits for word or 32 bits for dword integer instruction) in the accumulator contain the correct results. The internal extra-precision bits as well as the sign bit of the accumulator are undefined. Consequently, there are restrictions for subsequent instructions that use the data in the accumulator register created from the previous logical instruction.

- Only logical and data move instructions are allowed to source the accumulator. Results of other instructions (e.g. arithmetic or shift) are undefined.
- When the accumulator is the source of a data move (mov or sel) instruction, the destination operand must be of integer type (e.g. no conversion to float) and this instruction cannot have satuation instruction modifier.

9. EU Programming Guide

9.1 Assembler Pragmas

9.1.1 Declarations

A register or a register region can be declared as a symbol using the following form

.**declare** <symbol> **Base=**RegFile RegBase {.SubRegBase} **ElementSize=**ElementSize {**SrcRegion=**DefaultSrcRegion} {**DstRegion=**DefaultDstRegion} {**Type=**DefaultType}

The register file, the base of the register origin and the element size (in unit of bytes) are the mandatory parameters for a declared register region. Optionally, the base of the sub-register address, the default source region, the default destination region and the default type can be provided in the declaration for the symbol.

For immediate register addressing mode, the declared symbol can be used in the following Cartesian form

<symbol>(RegOff, SubRegOff) ← RegNum = *RegBase* + **RegOff***;* SubRegNum = *SubRegBase* + **SubRegOff**

or in the following simplified row-aligned form

<symbol>(RegOff) ← RegNum = *RegBase* + **RegOff;** SubRegNum = *SubRegBase*

For register-indirect-register-addressing mode, the declared symbol can be used to provide immediate address term in the following Cartesian form

<symbol>[IdxReg, RegOff, SubRegOff] ← RegNum (byte-aligned) = **[IdxReg]** +(*RegBase* + **RegOff**)*32 + (*SubRegBase* + **SubRegOff**)**ElementSize*

or in the following simplified row-aligned form

<symbol>[IdxReg, RegOff] ← RegNum (byte-aligned) = **[IdxReg]** +(*RegBase* + **RegOff**)*32

or in the form without the immediate address term

<symbol>[IdxReg] ← RegNum (byte-aligned) = **[IdxReg]** + *RegBase*

9.1.2 Defaults and Defines

The default execution size is set according to the destination register type as the following

Destination Register Type	Default Execution Size
UB | B	(16)
UW | W	(16)
F | UD | D	(8)

The default execution size can be overwritten globally for all instructions using

.**default_execution_size**	*(Execution_Size)*

or be set according the **destination** register type using

.**default_execution_size_*Type*** *(Execution_Size)*

The default register type can be set for all register files using

.**default_register_type**	*Type*

or be set per register file using

.**default_register_type_*RegFile***	*Type*

The default **source** register region for all symbols can be set using

.**default_source_register_region**	*<VirtStride; Width, HorzStride>*

or be set per register type using

.**default_source_register_region_*type***	*<VirtStride; Width, HorzStride>*

The default **destination** register region for all symbols can be set using

.**default_destination_register_region**	*< HorzStride>*

or be set per register type using

.**default_destination_register_region_*type***	*< HorzStride>*

Finally, the precompiler supports the string replacement statement of .define in the following form

.**define** <symbol>	Expression

Notes:

- **.declare** does not support nesting. In other words, each symbol in .declare must be self defined. This would allow the pre-processor to expand all symbols in one pass.
- **.define** does support nesting. Only string substitution is supported (currently).
- White space within square, angle and round brackets are allowed for easy source code alignment.

9.1.3 Example Pragma Usages

Example 1: Declaration for 8x4=32-Byte Regions:

The following symbol *Block* can be used to address any 8x4 byte region within the Cartisian system of a 16x8 byte GRF register area starting from r0.

Declaration

```
// 32x4 Byte Array

.declare Block   Base=r0 ElementSize=1 Region=<32;8,1> Type=b
```

Fully-Expressed Instr

```
mov(32)                ?:b                r0.16<32;8,1>:b          // r0
xxxxxxxxxxxxxxxxoooooooooxxxxxxxx

        // r1 xxxxxxxxxxxxxxxxoooooooooxxxxxxxx

        // r2 xxxxxxxxxxxxxxxxoooooooooxxxxxxxx

        // r3 xxxxxxxxxxxxxxxxoooooooooxxxxxxxx
```

Short-handed Instr

```
Mov            ?:b            Block(0,16)                                    //
(0,16): RegNum=0, SubRegNum=16
```

Example 2: Declaration for 8x1 Float Regions:

The following symbol *Trans* can be used to address any 8x1 float region within the Cartisian system of a 8x4 float GRF register area starting from r5.

Declaration

```
// 8x4 float Array starting at r5

.declare Trans Base=r5 ElementSize=4 Region=<0;8,1> Type=f
```

Fully-Expressed Instr

```
mov(8)          ?:f             r6.0<0;8,1>:f                   // 2nd 16x1 Row of
Trans. Matrix

        // r5 FFFFFFFF

        // r6 OOOOOOOO

        // r7 FFFFFFFF

        // r8 FFFFFFFF
```

Short-handed Instr

```
mov             ?:f             Trans(1)                        //
RegNum = 5+1 = 6
```

Example 3: Declaration for 8x1 Float Regions with 1x1 Indirect Addressing:

Trans region defined (same as in the previous example) is used in conjunction with the address register.

Declaration

```
//8x4 float data array and 16x1 word address array

.declare Trans   Base=r5 ElementSize=4 Region=<0;8,1> Type=f
```

Fully-Expressed Instr

```
mov(8)          ?:f             r[a0.0,224]<0;8,1>:f
```

Short-handed Instr

```
mov             ?:f             Trans[a0.0,2]                   // [a0.0 +
5*32 + 2*32]
```

Example 4: Declaration with VxH Indirect Addressing:

The VxH register-indirect-register-addressing for Trans can be provided in the following short-hand form.

Declaration	//8x4 float data array and word indices .declare Trans Base=r5 ElementSize=4 Region=<0;8,1> Type=f
Fully-Expressed Instr	mov(8) ?:f r[a0.0,224]<1,0>:f
Short-handed Instr	mov ?:f Trans[a0.0,2]<1,0> // [a0.0+224] [a0.1+224] … [a0.7+224]

Example 5: Declaration with Vx1 Indirect Addressing:

As width (4) is smaller than the execution region size (8), multiple indexed registers are used.

Declaration	//8x4 float data array and word address array .declare Trans Base=r5 ElementSize=4 Region=<0;8,1> Type=f
Fully-Expressed Instr	mov(8) ?:f r[a0.0,244]<4,1>:f
Short-handed Instr	mov ?:f Trans[a0.0,2]<4,1> // [a0.0+224] [a0.1+224]

9.1.4 Assembly Programming Guideline

The following program skeleton illustrates the basic structure of a typical assembly program.

```
//          single line comment


/*

      block comment

*/


<preproc_directive>      // macros, include, etc.  Are global – handled by the pre-processor

<preproc_directive>      // applies to all code that follows in sequence


// ------------ some kernel ---------------------------

.kernel <kernel_name_string>    // [REQUIRED]

                                                                    // ------- Register requirements --------

 .reg_count_total     <uint>         // [REQUIRED] a more direct way to specify the exact parameters
require

 .reg_count_payload   <uint>      // [REQUIRED] rather than to have to indirectly do that by adding the

                                                                    //   the payload and temps together to
get the total (as is the case now)

                                                                    // Note: no more “reg-count-temp”


                                                                    // -------------- Defaults ---------------

 <default…>                                                 // these should be specified per-kernel and have
only kernel-scope

 <default…>                                                 // Same defaults as those already defined in the
ISA doc, but just

 <default…>                                                 // moved within the kernel to make each kernel
completely self-sufficient

                                                                    // and not impacted defaults of earlier
kernels
```

```
                                                    // --------- Memory Requirements ---------
                                                    // [optional] memory block info (just a
placeholder for now...)
 <MBDa>                                             //     memory block descriptor a (TBD)
 <MBDb>                                             //     memory block descriptor b (TBD)
 <MBDc>                                             //     memory block descriptor c (TBD)
 <MBDd>                                             //     memory block descriptor d (TBD)

                                                    // ---------------- Code  ----------------
 .code                                          // [REQUIRED]
                 <instruction>
                 <instruction>
                 <instruction>
          <LabelLine>                           // labels are code-block scope
                 <instruction>
                 <instruction>

 .end_code                                      // [REQUIRED]

.end_kernel                                     // [REQUIRED]

// --------- next kernel -------------

// --------- next kernel -------------

// ...
```

9.2 Usage Examples

9.2.1 Vector Immediate

The immediate form of vector allows a constant vector to be in-lined in the instruction stream. An immediate vector is denoted by type v as *imm32:v*, where the 32-bit immediate field is partitioned into 8 4-bit subfields. Each 4-bit subfield contains a signed integer value in 2's compliment form. Therefore each 4-bit subfield has a range of [-8, +7]. This is depicted in the following figure.

31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
V7	V6	V5	V4	V3	V2	V1	V0

9.2.1.1 Supporting OpenGL Vertex Shader Instruction SWZ

When an OpenGL Vertex Shader program is converted to run in Vertex Pair, i.e. two 4-wide vectors in parallel, the special OpenGL Shader instruction SWZ (Swizzle) needs to be emulated. OpenGL SWZ instruction uses an extended swizzle control field that, in addition to the 4-wide full swizzle control, also includes constant 0 and 1 replacement as well as per channel sign reversal. The later two are not supported by the native instruction. The vector immediate can significantly reduce the overhead of emulating such OpenGL instruction.

Consider an OpenGL Shader instruction in the form of

```
SWZ     r1      r0.0-zx-1       // Expected results: r1.x = 0; r1.y = -r0.z; r1.z = r0.x; r1.w = -1
```

It can be emulated by the following three instructions.

```
mul     (8)     r1.0<1>:f       r0.xzxz         0x1F111F11:v   // Constant vector of (1 -1 1 1 1 -1 1 1)

mov (1) f0.0                    8b'10011001                    // Set flag & masked out channels y and z

(f0.0)mov(8) r1.0<1>:f          0x000F000F:v                   // Constant vector of (0 0 0 -1 0 0 0 -1)
```

In case that only 0, 1, -1 channel replacement is used and there is no signed swizzle, it may be emulated in two instructions. This is illustrated by the following example:

OpenGL:

```
SWZ     r1      r0.0zx-1        // Expected results: r1.x = 0; r1.y = r0.z; r1.z = r0.x; r1.w = -1
```

DevSNB:

```
mov (1) f0.0        8b'01100110                    // Set flag and masked out channels x and w

(f0.0)sel (8) r1.0<1>:f r0.yzxy     0x000F000F:v   // Constant vector of (0 0 0 -1 0 0 0 -1)
```

9.2.2 Destination Mask for DP4 and Destination Dependency Control

The following example demonstrates the use of destination mask mode of floating point dot-product instruction as well as the use of destination dependency control to improve performance (i.e., avoiding unnecessary thread switch due to possible false dependencies).

With destination dependency control, the DevSNB instructions are as the following. The first instruction in the sequence checks for the destination dependency, but does not clear the dependency bit. The subsequent two instructions would do neither of them. The last instruction avoids checking the destination dependency, but at completion, it clears the destination scoreboard. It ensures that the content of the destination register is coherent, if any of the following instructions uses the same register as source.

```
dp4 (8) r5.0<1>.x:f r0.0<4;4,1>:f r4.0<4;4,1>:f     {NoDDClr}

dp4 (8) r5.0<1>.y:f r1.0<4;4,1>:f r4.0<4;4,1>:f     {NoDDClr, NoDDCChk}

dp4 (8) r5.0<1>.z:f r2.0<4;4,1>:f r4.0<4;4,1>:f     {NoDDClr, NoDDCChk}

dp4 (8) r5.0<1>.w:f r3.0<4;4,1>:f r4.0<4;4,1>:f     {NoDDChk}
```

Just as a comparison, IF DP4 implies reduction at the destination; additional shifted moves are required to achieve the same results. The corresponding codes are as the following. The lower performance due to the additional three move instruction as well as added back-to-back dependencies shows that why we choose to implement the destination channel replication for floating point DP4.

```
dp4 (8) r5.0<1>.y:f      r1.0<4;4,1>:f     r4.0<4;4,1>:f

mov (1) r5.1<1>:f        r8.0<1;1,1>:f

dp4 (8) r5.0<1>.z:f      r2.0<4;4,1>:f     r4.0<4;4,1>:f

mov (1) r5.2<1>:f        r8.0<1;1,1>:f

dp4 (8) r5.0<1>.w:f      r3.0<4;4,1>:f     r4.0<4;4,1>:f

mov (1) r5.3<1>:f        r8.0<1;1,1>:f

dp4 (8) r5.0<1>.x:f      r0.0<4;4,1>:f     r4.0<4;4,1>:f
```

9.2.3 Null Register as the Destination

Null register can be used as the destination for most of the instructions. Here are some example usages.

- Null as destination for regular ALU instructions: As all ALU instructions can be configured to update the flag registers using the conditional modifiers, it is not necessary to have a destination register if the programmer only cares about the conditionals of the operation. In that case, a null in the destination operand field saves register space as well as one less dependency checking.
- Null as the destination for SEND/STOR instructions: for the send instruction that only send messages out to an external unit and does not require any return data or feedback, a null in the destination register field signifies the case.

9.2.4 Use of LINE Instruction

LINE instruction is specifically designed to speed up floating point vector/matrix computation when a program operates in channel serial.

The following example is to compute homogeneous dot product for OpenGL pixel shader running in Channel Serial. In this example, an original OpenGL PS instruction is like

dph R2.x R0 R1

With register remapping, we can store the input coefficient vector R0 in original format in r0, but 8 sets of input coordinate vectors in channel serial format in r2, r3, r4 and r5, and the destination R2.x component in r6.

r0: Cx0 Cy0 Cz0 Co0 DC DC DC DC

r2: X0 X1 … X7

r3: Y0 Y1 … Y7

r4: Z0 Z1 … Z7

r5: W0 W1 … W7

The objective is to compute the following DPH equations and store the results in r6 as

R6: (X0*Cx0+Y0*Cy0+Z0*Cz0+Co0) ... (X7*Cx0+Y7*Cy0+Z7*Cz0+Co0)

Example 9-1. Homogeneous Dot Product in Channel Serial

```
//-----------------------------------------------------------------

// Example compute homogeneous dot product in channel serial scenario

//-----------------------------------------------------------------


line (8) acc:f   r0<0;1,0>:f   r2<0;8,1>:f          // does acc = X# * Cx0 + Co0

mac  (8) acc:f   r0.1<0;1,0>:f r3<0;8,1>:f          // does acc.# = Y# * Cy0 + acc.#

mac  (8) r6<1>:f r0.2<0;1,0>:f r4<0;8,1>:f                // does r6.# = Z# * Cz0 + acc.#
```

9.2.5 Mask for SEND Instruction

Execution mask (upto 16 bits) for the SEND instruction is transferred to the Shared Function.

9.2.5.1 Channel Enables for Extended Math Unit

The following example demonstrates how to use the SEND instruction to get service from the Extended Math unit.

Let's consider COS instruction in the following form

[([!]p0.{select|any|all})] cos[_sat] dest[.mask], [-]src0[_abs][.swizzle]

For a SIMD4x2 VS implementation with the following register mappings

p0 → f0.0

src0 → r0

dest → r1

The equivalent DevSNB instruction is as the following

[([!]f0.0.{select|any4h|all4h})] SEND (8) r1[.mask]:f m0 [-][(abs)]r0[.swizzle]:f MATHBOX|COS[|SAT]

If the source swizzle is replication, the message description field can be modified to MATHBOX|COS|SCALAR to take advantage of the fast mode (scalar mode) supported by the Extended Math. The implied move of the SEND instruction is equivalent to the following instruction:

MOV (8) m0[.mask]:f [-][(abs)]r0.0[.swizzle]:f {NoMask}

For a SIMD16 PS implementation, the register mappings are as the followings

p0 → f0…f3 // in order of R, G, B, A

src0 → r0,r1; r2,r3; r4,r5; r6,r7

dest → r8,r9; r10,r11; r12,r13; r14,r15

There are several ways to translate the instruction, depending on the operand/instruction modifiers present in the instruction. If predicate is not present and the source swizzle is replication, say, src0.y, which is r2-r3, the translation could be as the following instructions

```
send (8) r8:f m0        -(abs)r2:f MATHBOX|COS
send (8) r9:f m1        -(abs)r3:f MATHBOX|COS {SecHalf}    // use the second half of 8 flag bits
mov (16) r10:f  r8:f                                        // All destination color chan's are same
mov (16) r12:f  r8:f                                        // MOV is faster than most MathBox func's
mov (16) r14:f  r8:f                                        // These MOV's are compressed instructions
```

Notice that instead of issuing Extended Math messages with the same input data, destination color channel replication is performed by the MOV instructions. This is faster for the thread for most cases as

many Extended Math functions consume multiple cycles. This also conserves message bus bandwidth as well as the usage of the shared resource – Extended Math. The destination mask in the instruction indicates which of the r8 to r15 registers are updated. If the source swizzle is not replication, there will be 8 SEND instructions.

With predication on, if the predication modifier is p0.select, translation is to take the selected flag register f#. The other predication modifiers '.any' and '.all' are translated into '.any4v' and '.all4v', respectively. Notice that with predication on, it is not required to run all 4 pixels in a subspan in the same way, so no need to enforce .any4h/.any4v. The following example shows the instruction with predication (but without .select modifier).

```
(f0[.any4v|.all4v]) send (8) r8:f m0        -(abs)r2:f MATHBOX|COS

(f0[.any4v|.all4v]) send (8) r9:f m1        -(abs)r3:f MATHBOX|COS {SecHalf}

(f1[.any4v|.all4v]) mov (16) r10:f r8:f             // All destination color chan's are same

(f2[.any4v|.all4v]) mov (16) r12:f r8:f             // MOV is faster than most MathBox func's

(f3[.any4v|.all4v]) mov (16) r14:f r8:f             // These MOV's are compressed instructions
```

The same instructions works also for predication with select component modifier. We simply replase f0 to f3 above by the selected flag register, say, f1. The modifier of any4h/all4v would also work.

9.2.5.2 Channel Enables for Scratch Memory

The following example demonstrates how to use the SEND instruction to get service from the Data Port for scratch memory access.

Let's consider the case when a scratch register is the destination of an instruction.

```
p0          →     f0

src0   →    r0

src1   →    r1

dest   →    s2 / r10
```

We have

```
add (8) m1:f [-][(abs)]r0[.swizzle]:f [-][(abs)]r1[.swizzle]:f

mov (8) r3:d r0:d {NoMask}      // move scratch base address to be assembled with offset values

mov (1) r3.0:d 2*32 {NoMask}    // s2 for vertex 0

mov (1) r3.1:d 2*32+16 {NoMask}         // s2 for vertex 1

    [([!]f0.{sel|any4h|all4h})] send (8) null[.mask] m0 r3 DATAPORT|RC|WRITE_SIMD2
```

Notice that with a null as the posted destination register, we are able to transfer the [.mask] over the message channel enables. In many cases for scratch memory assess, a write-with-commit is required, therefore, the posted destination register could be r10.

Now, let's consider the PS case when a scratch register is the destination of an instruction.

p0 → f0-f4

src0 → r0-r7

src1 → r8-r15

dest → s16-s23 / r16-r23

When predication is not on (or predication with swizzle control on), we have

add (16) m4:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

add (16) m6:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

add (16) m8:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

add (16) m10:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

mov (8) r3:d 0x76543210:v {NoMask} // ramp function

mul (16) acc0:d r3:d 16 {NoMask} // ramp function

add (8) acc0:d acc0:d 64 {NoMask,SecHalf} // ramp function

add (16) m2:d acc0:d 2*256 {NoMask} // ramp function

send (16) **null** m1 r3 DATAPORT|RC|WRITE_SIMD16

As there is no bit left from the unit specified descriptor field, the 4 bit mask must be put into the header field in m1, which requires at least two more instructions.

Alternatively, or for the case that predication without modifier is on, we can do a read-modify-write.

mov (8) r3:d 0x76543210:v {NoMask} // ramp function

mul (16) acc0:d r3:d 16 {NoMask} // ramp function

add (8) acc0:d acc0:d 64 {NoMask,SecHalf} // ramp function

add (16) m2:d acc0:d 2*256 {NoMask} // ramp function

send (16) **r16** m1 r3 DATAPORT|RC|READ_SIMD16 // read from scratch

// some of the following four instructions may be omitted based on [.mask] field

[([!]f0.{sel|any4v|all4v})] add (16) r16:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

[([!]f0.{sel|any4v|all4v})] add (16) r18:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

[([!]f0.{sel|any4v|all4v})] add (16) r20:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

[([!]f0.{sel|any4v|all4v})] add (16) r22:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

mov (16) m4:f r16:f {NoMask}

```
mov (16) m6:f r18:f {NoMask}

mov (16) m8:f r20:f {NoMask}

mov (16) m10:f r22:f {NoMask}

    send (16) null m1 null DATAPORT|RC|WRITE_SIMD16 {NoMask}     // write back to scratch
```

9.2.6 Flow Control Instructions

Unconditional branches are performed through direct manipulation of the 32-bit IP architectural register. For example:

```
        mov (1) IP <memory_address>                 // jump absolute
        add (1) IP  IP  <byte_count>  // jump relative
```

Note that jump distances are specified in terms of bytes, as opposed to instruction counts in the case of *break*, *halt*, etc. To minimize confusion, an assembler-only instruction 'jmp <inst_count>', where <inst_count> is an immediate term, may be defined which takes an instruction count for a distance. The *jmp* pseudo-opcode can be mapped to an "add (1) ip ip <inst_count> * 16" instruction.

Also note that IP is always an instruction-sized aligned address (16 bytes), thus the 4 LSB's are not maintained in the IP architectural register and should not be relied upon by software.

IP, when used as a source operand, reflects the memory address of the instruction in which it is used. The following are examples illustrating the use of IP:

```
            add (1) IP      4*16            // jumps to HERE_1
            add (1) IP      0x35            // jumps to HERE_1 (4 lsb's don't-care)
            <instruction>
            <instruction>
HERE_1:         <instruction>
HERE_2:         <instruction>
            <instruction>
            add (1) IP -2*16                // jumps to HERE_2
            ...
            add (1) IP 0            // infinite loop
            add (1) IP 0xF          // infinite loop
            ...
```

Note for Assembler: The `if/iff/else/while/break instructions identify relative addresses as the targets of an implicit jump associated with the instruction. These are optional in the assembly syntax as the jitter can determine the location of the matching instruction (e.g. matching endif instruction for a given if instruction).`

9.2.7 Execution Masking

9.2.7.1 Branching

Example 9-2. If / Else / EndIf

```
//-----------------------------------------------------------------
// Example if/else/endif scenario
//          "if (r5==r4) ...else ... end-if"
//-----------------------------------------------------------------
                ...
                cmp.e.f0 (8)  null r5 r4    // does r5 == r4?
                (f0) if (8)   HERE_1                // "if" part - save then update IMASK;
                                                    //    or goto the 'else' if all false
                ...
                ...
HERE_1:                                                     // now do the 'else' part
                else (8) HERE_2                     // "else" part - invert IMASK
                                                    //    or goto the 'endif' if all false
                ...
                ...
HERE_2:
                endif                       // "end-if" part – restore IMASK
                ....                        // and continue...
```

If it is known that the code has no nested conditionals, a predicate can be used for a lower overhead, more efficient if/else/endif. (One must consider the probability of all channels taking the same branch, and the number of instructions under the if/else blocks as to which conditional method, predicate or mask, is most efficient).

9.2.7.2 Fast-If

Below is an example of a fast-if instruction. For the 'iff' instruction, only and iff-endif construct is allowed, as opposed to a if-else-endif. Note that the target address for branching if all enabled channels fail is one instruction beyond the endif, as the 'iff' does not push and update the IMask unless the branch is taken for at least one execution channel.

Example 9-3. Fast If

```
//-----------------------------------------------------------------
// Example – Fast If
//          One instruction overhead conditional
//-----------------------------------------------------------------
                ...
                cmp.e.f0 (8)  null r5 r4            // any flag update
                ...
        (f0)    iff (8) HERE_1                      // "fast-if" – only pushes IMask;
                                                            //  if execution falls through,
                                                            //  else go to HERE_1
                ...
                ...
                endif                               // "end-if" part – restores IMask
HERE_1:
                    ...                             // and continue...
```

9.2.7.3 Cascade Branching

As there is no 'elseif' instruction, a C-like cascade branching such as if / elseif / else / endif, can be realized using the basic building blocks of if / else / endif as shown in the following example. Notice that two 'endif's' are required in order to pop the IStack correctly.

Example 9-4 If / Elseif / Else / EndIf

```
//-----------------------------------------------------------------
// Example if/elseif/else/endif scenario
//          "if (r5==r4) ...elseif (r6>r7) else ... end-if"
//-----------------------------------------------------------------
                ...
                cmp.e.f0 (8)  null r5 r4    // does r5 == r4?
        (f0)    if (8) HERE_1               // "if" part - save then update IMask;
                                                    //  or go to the 'else' part if all false
                ...
```

```
                ...
HERE_1:                                                 // now do the 'else' part
                else (8) HERE_2                 // "else if" part - invert IMask
                                                //    or go to the 'else' part if all false
                cmp.g.f0 (8)  null r6 r7    // is r6 > r7?
        (f0)    if (8) HERE_3               // "if" part - save then update IMask;
                                                //    or go to the 'else' part if all false
                ...
                ...
HERE_3:                                                 // now do the 'else' part
                else (8) HERE_4                 // "else" part - invert IMask
                                                //    or go to the 'end-if' part if all false
                ...
                ...
HERE_4:
                endif                           // "end-if" part – restore IMask for elseif
HERE_2:
                endif                           // "end-if" part – restore IMask for if
                ....
```

9.2.7.4 Compound Branches

Compound branches are supported through the ability logically combine flag registers for each intermediate result.

Example 9-5 Compound Branch

```
//-----------------------------------------------------------------
// Example:  "if (r0 > r1) OR (r2 <= r3)"
//-----------------------------------------------------------------
                ...
                cmp.g.f0 (8)  null r0:d  r1:d           // r0 > r1?
                cmp.le.f1 (8) null r2:d  r3:d           // r2 <= r3?
```

```
                or (1) f0:w f0:w  f1:w                  // combine f0 and f1
        (f0) if (8) HERE_1                              // Can now do normal if/else
                ...
                ...
HERE_1:         endif

                ...
```

Example 9-6. Compound Branch Using 'Any' or 'All'

```
//-----------------------------------------------------------------
// Example:  assuming we're doing a channel-serial vector in r0-r3
//           We want to know if all components of the vector are > 0x80
//-----------------------------------------------------------------
                ...
                cmp.g.f0 (16)   null r0 0x80            // r0 > 0x80?
                cmp.g.f1 (16)   null r1 0x80            // r1 > 0x80?
                cmp.g.f2 (16)   null r2 0x80            // r0 > 0x80?
                cmp.g.f3 (16)   null r3 0x80            // r1 > 0x80?
        (f0.all4v) if (16) HERE_1
                ...
                ...                     // code executed only for those channels
                ...                     // where per-channel r0,r1,r2,r3 all > 0x80
                ...
HERE_1:         endif
                ...                                     //  and continue...
```

9.2.7.5 Looping

Due to DevSNB's SIMD-16 architecture, it must support the case of up to 16 loops running in parallel. These must be handled as independent loops, each with its own loop-exit condition which could occur after a different number of loop iterations. To account for each channel's progress, a 16b loop-mask 'LMask' is defined with 1b associated to each execution channel. This mask keeps track of which channels remain active inside a loop block.

Basic Do-While Loop

Example 9-7 illustrates the most basic loop. Two operations must be accomplished before loop entry. (1) Prior to loop entry, there is some subset of enabled channels as dictated by the code sequence prior. In general, the active status of each channel is indicated in the virtual EMask any point in time. These active channels will become the channels over which the loop is run, and LMask must be initialized with the EMask value. (2) Since a given loop may be nested within another loop, the previous LMask & CMask must be saved to the LStack for later restoration upon loop completion. The 'msave' instruction performs both the save and update in a single instruction, and thus all loop-blocks should be fronted with a "msave LStack LMask" and "msave LStack CMask" operation.

Note that the LMask and CMask share the same mask-stack. Thus, CMask must always be a 1's-subset of the LMask for proper stack operation. This is the case if CMask is updated to LMask each pass through the loop (see Example 9-7) and through the 'break' instruction updating both masks.

Each pass through the loop, a loop terminating operation must be evaluated and stored in a flag register. This condition must be evaluated on a channel-by-channel basis as exemplified:

```
cmp.z.f0        (8) null r2 d3                  // any operation that updates a flag
```

The result of this operation sets a bit per channel in the specified flag register, which is then used in the 'while' instruction. As loops are performed, channels may become disabled as their termination condition is met.

'While' termination is determined on a channel-by-channel basis by the logical AND of corresponding bit positions of AMask, CMask and the specified flag. If the result is '1' the channel remains enabled for the next pass of the loop; if '0' the channel is disabled until loop fall-through. The 'while' instruction causes the LMask to be updated with the latest result of enabled channels. If any channel remains enabled (LMask != ...000b), an additional pass through the loop is made. Once a channel is terminated for the loop operation, it remains terminated until the loop is complete for all channels.

Upon fall through, the 'while' instruction causes the previously saved LMask & CMask to be popped from the LStack, enabling execution on the same subset of channels enabled prior to loop entry (unless a channel had been otherwise terminate inside the loop via 'halt').

Example 9-7. Basic Loop Construct

```
//-----------------------------------------------
//      Example: Basic do-while loop structure
//-----------------------------------------------
                ...
                do                              // save L/CMask & update
BEGIN_LOOP:
                mov (1) CMask LMask   {NoMask}  // update CMask for this pass
                ...
                ...
                <some flag update>
```

```
(<p>)   while (8)  BEGIN_LOOP                    // cond. branch
                                                 //    + restores LMask on fall-thru
        ...
```

Do-While Loop with Break

A loop may also be terminated for any channel via the 'break' instruction. The 'break' instruction causes the corresponding bit positions of enabled channels to be cleared in the LMask. If the updated LMask = ...000b, a branch is made to the specified instruction location. An example is shown below in which the 'break' is at the same conditional-nesting level as the terminating 'while'. Its primary value may simply be to support a "do...break.. while (true)" –type structure for a more direct 1:1 translation from higher-level source code.

Example 9-8. Loop Construct With Non-Nested 'Break'

```
//-------------------------------------------------------------
//          Example: While-true loop
//-------------------------------------------------------------
#define BrkCode(i,d)     (i << 16) + d


                         do                                  // save L/CMask & update
BEGIN_LOOP:
                         mov (1) CMask LMask   {NoMask} // update CMask for this pass
                         ...
                         <some flag update>
                (<p>)    break (8) BrkCode(0,HERE_1)   // Restores LMask when all
                                                       // channels complete loop.
                         ...
                         ...
                         while (8) BEGIN_LOOP          // while true
HERE_1:
                         ...
```

A break condition may occur from various levels of nested-ifs. This gives rise to the possibility that a the loop may terminate from within nested 'if's, and due to the jump inherent in the 'break' instruction, the associated 'endif's are not encountered to clean-up the IStack as nesting levels are exited.

Example 9-9 Loop Construct With 'Break' From Within Nested If's

```
//-------------------------------------------------------------
//          Example: General Loop Structure w/ break inside if's
//-------------------------------------------------------------
#define BrkCode(i,d)      (i << 16) + d

                          do                                        // save L/CMask & update
BEGIN_LOOP:
                          mov (1) CMask LMask   {NoMask} // update CMask for this pass
                          ...
                          if ...
                          if ...
                          if ...
                          ...
                  (<p>)   break (8) BrkCode(3,HERE_1)   // we're 3 levels deep, so
                          ...
                          endif
                          endif
                          endif
                          ...
                  (<p>)   break (8) BrkCode(0,HERE_1)
                          ...
                          while (8) <flag_spec> BEGIN_LOOP      // cond. branch
                                                                // + restores C/LMask on fall-thru
HERE_1:
```

Do-While Loop with Continue

A continue instruction ‘cont’ is provided skip to the next iteration of the loop. Because not all channels participating in the loop may be enabled at the time this instruction is executed, some channels may require continuation of the loop. A special mask ‘CMask’ is defined which accounts for channels temporarily disabled for the current loop pass.

Since loops may nested, the CMask must be saved and restored around a loop similar to LMask. Since the CMask value within a properly constructed loop is always a subset of the LMask, it can share the LStack for storage, so long as it is pushed after LMask as shown in Example 9-10. This save/restore operations are not required if the loop being entered does not have any occurrence of a continue instruction.

Example 9-10. Do-While with Continue

```
//-------------------------------------------------------------

//          Example: General Loop Structure w/ basic break and cont.

//-------------------------------------------------------------

#define ContCode(i,d)    (i << 16) + d


                         do                          // save L/CMask & update

BEGIN_LOOP:

                         mov (1) CMask EMask         // re-initialize CMask for this pass

                         ...

                         ...

                 (<p>) cont (8) ContCode(0,HERE_1)

                         ...

HERE_1:

                 (<p>)   while (8) BEGIN_LOOP        // cond. branch

                                                     //   + restores C/LMask on fall-thru

                 ...
```

9.2.7.6 Indexed Jump

Example 9-11. Indexed Jump

```
//-----------------------------------------------------------------
// Code example shows the use of jmpi to perform a case statement
// of any number of options in 3 jumps
//-----------------------------------------------------------------
.default_execution_size 8
...
jmpi r0<0,1,0>              // jump relative, based on r0.a.x
                            // ----- Jump Table ------
jmp HERE_0                  // redirect for case 0
jmp HERE_1                  // redirect for case 1
jmp HERE_2                  // redirect for case 2
jmp HERE_3                  // redirect for case 3
...
HERE_0:                     // ... case 0 ...
...
jmp DONE
HERE_1:                     // ... case 1 ...
...
jmp DONE
HERE_2:                     // ... case 2 ...
...
jmp DONE
HERE_3:                     // ... case 3 ...
...
DONE:
...                         //  and continue...
```

Revision History

Revision Number	Description	Revision Date
1.1	Edited extraneous content	May 2011
1.0	First 2011 OpenSource edition	May 2011

§§