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

Contents

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.

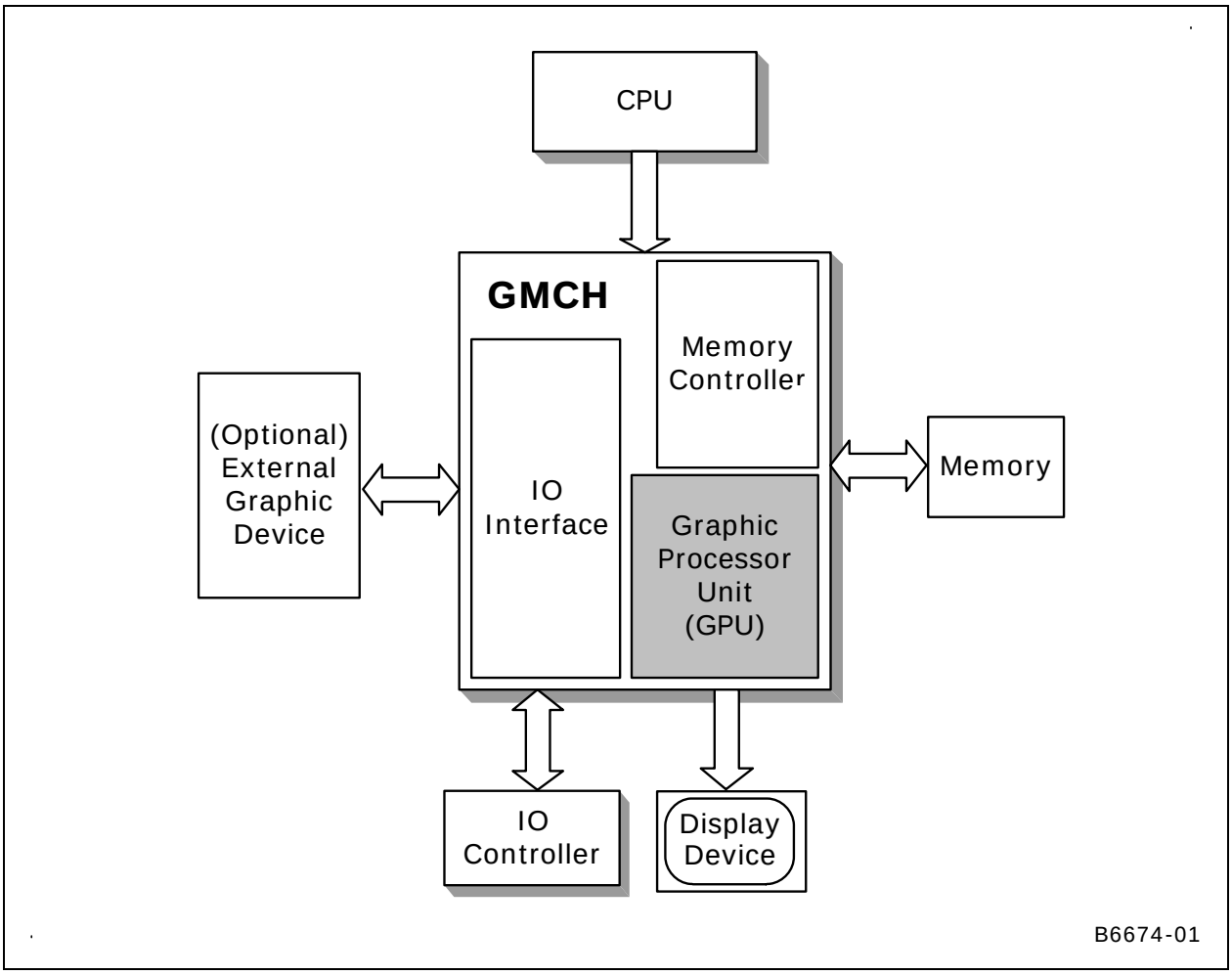
2. Graphics Device Overview

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.

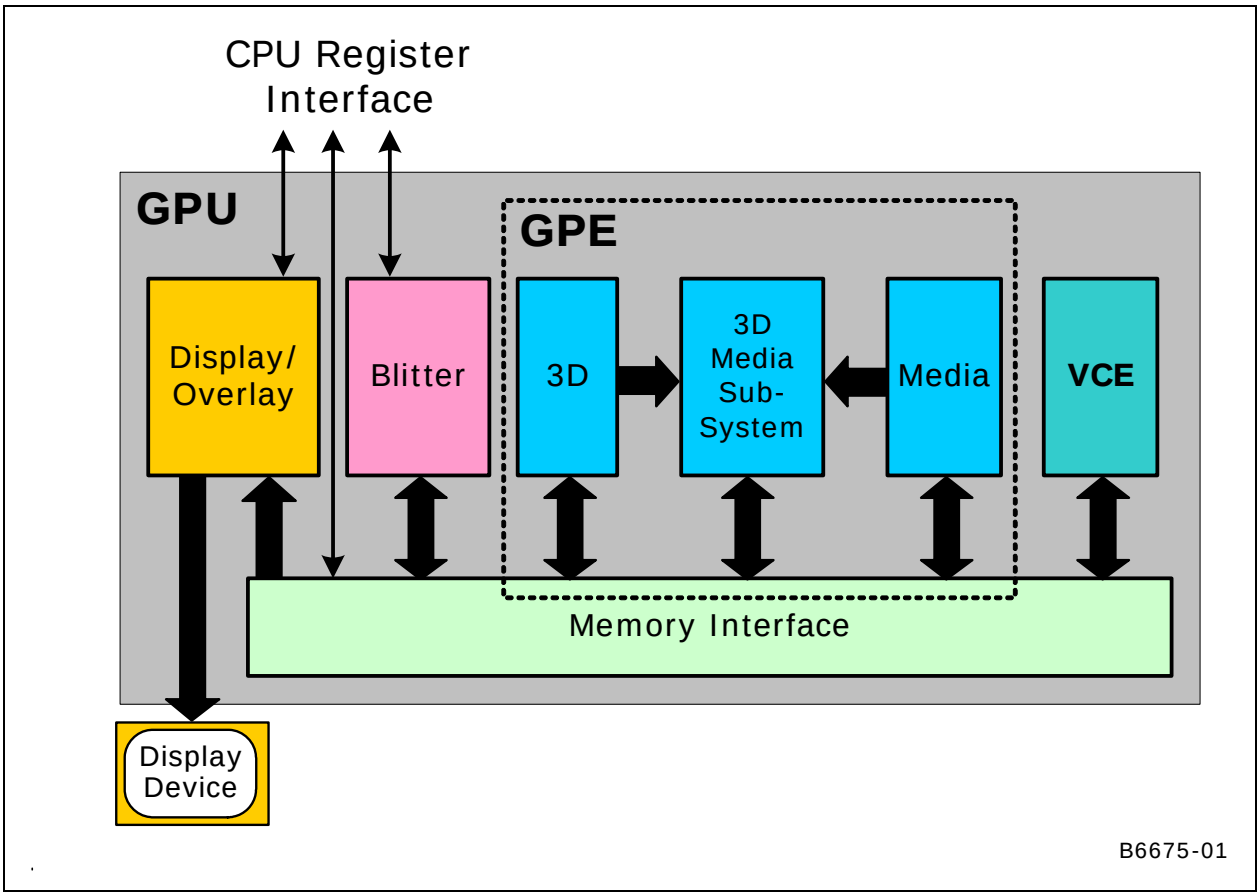


Figure 2-2. Block Diagram of the GPU

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.

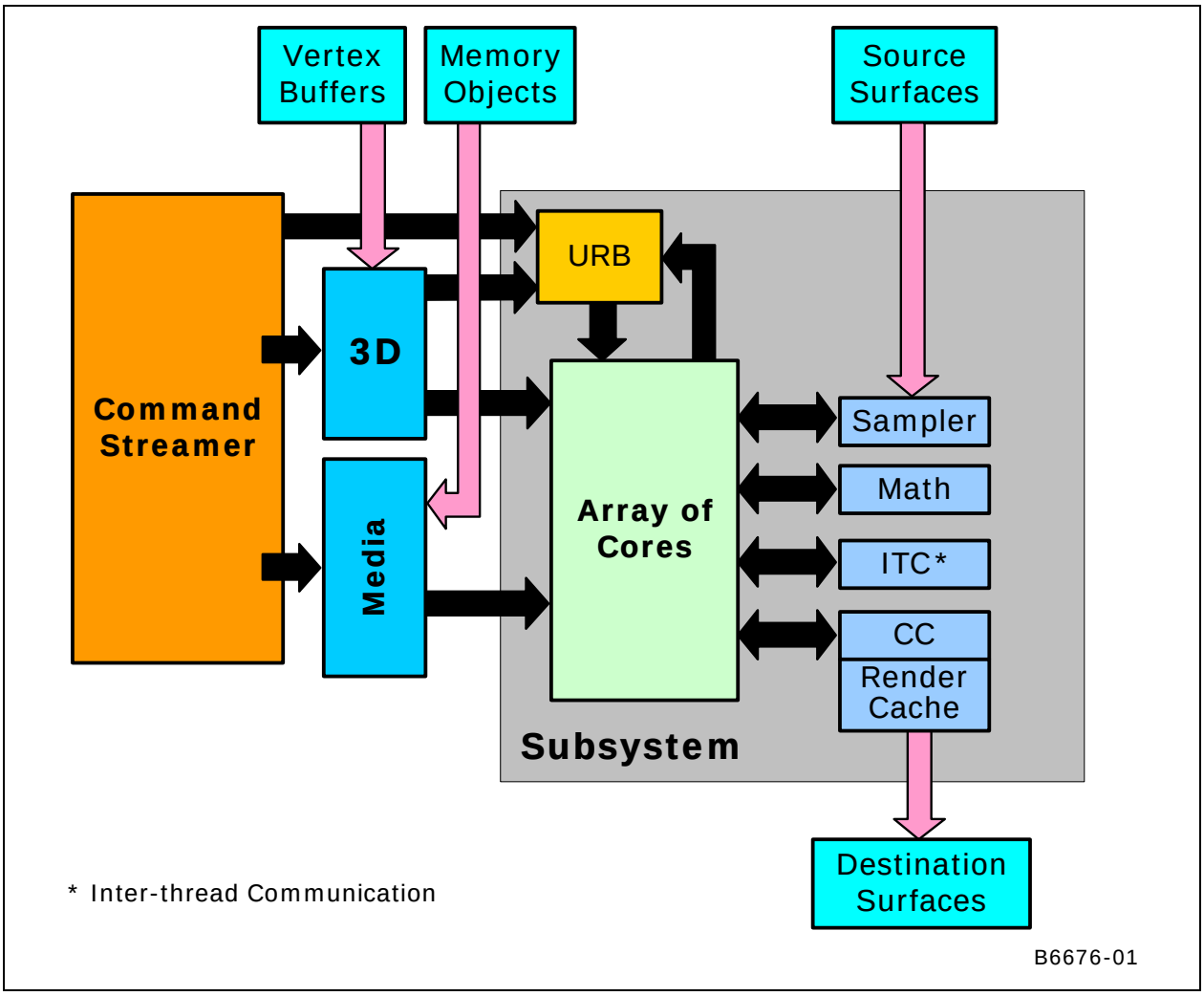
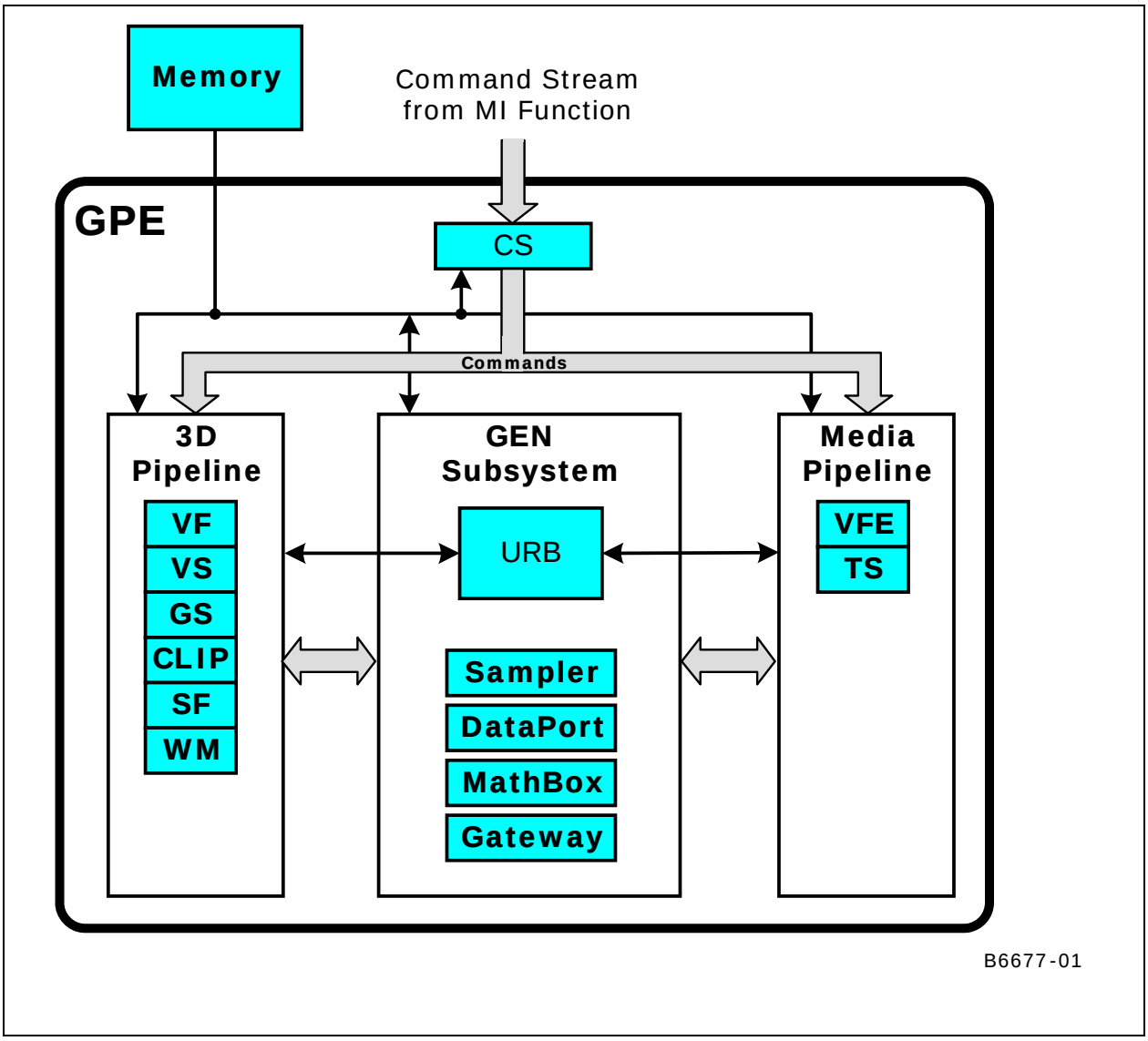


Figure 3-1. The Graphics Processing Engine

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

	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

	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

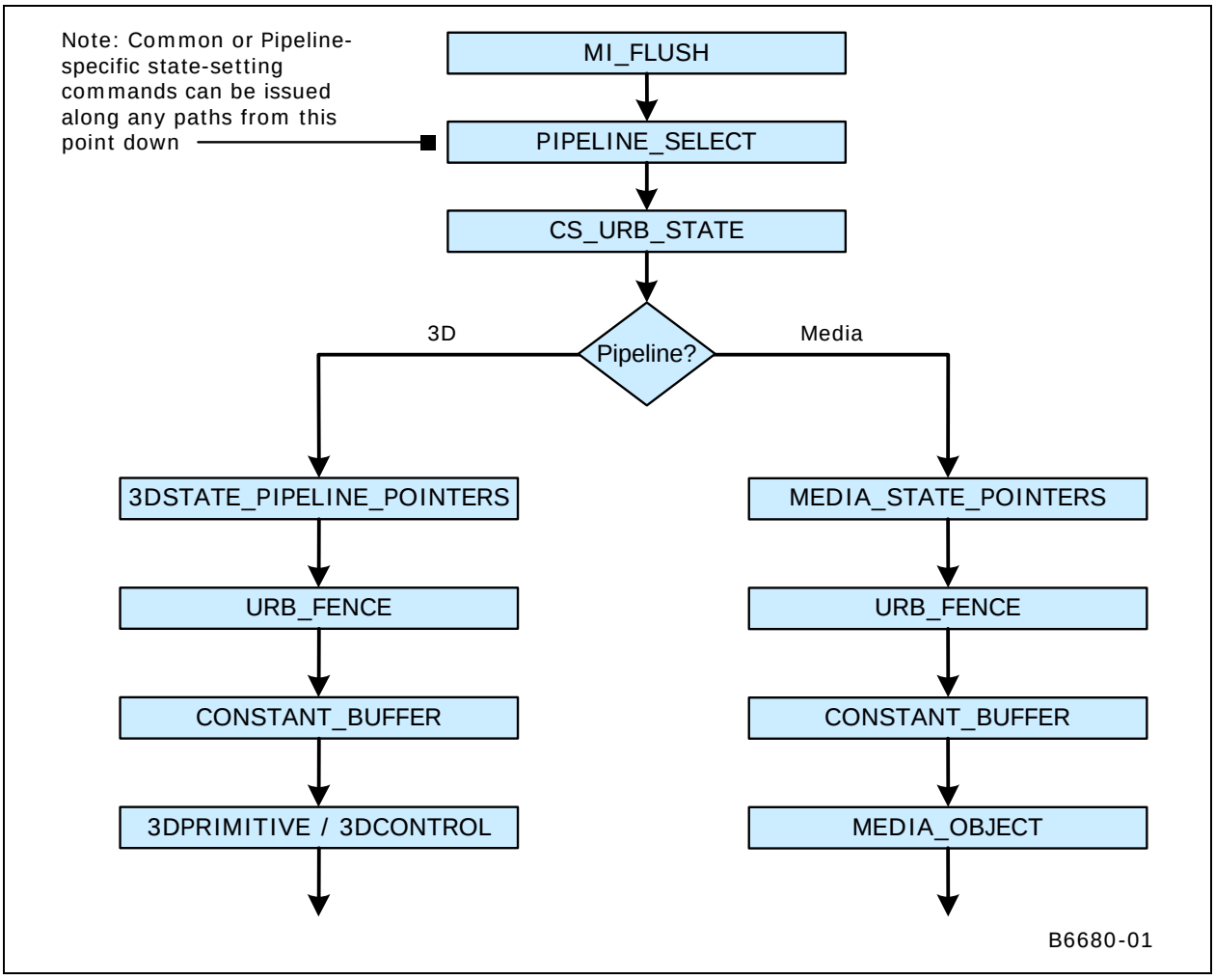
STATE_SIP			
Project:	All	**Length Bias:**	2
The STATE_SIP command specifies the starting instruction location of the System Routine that is shared by all threads in execution.			

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: 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,

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

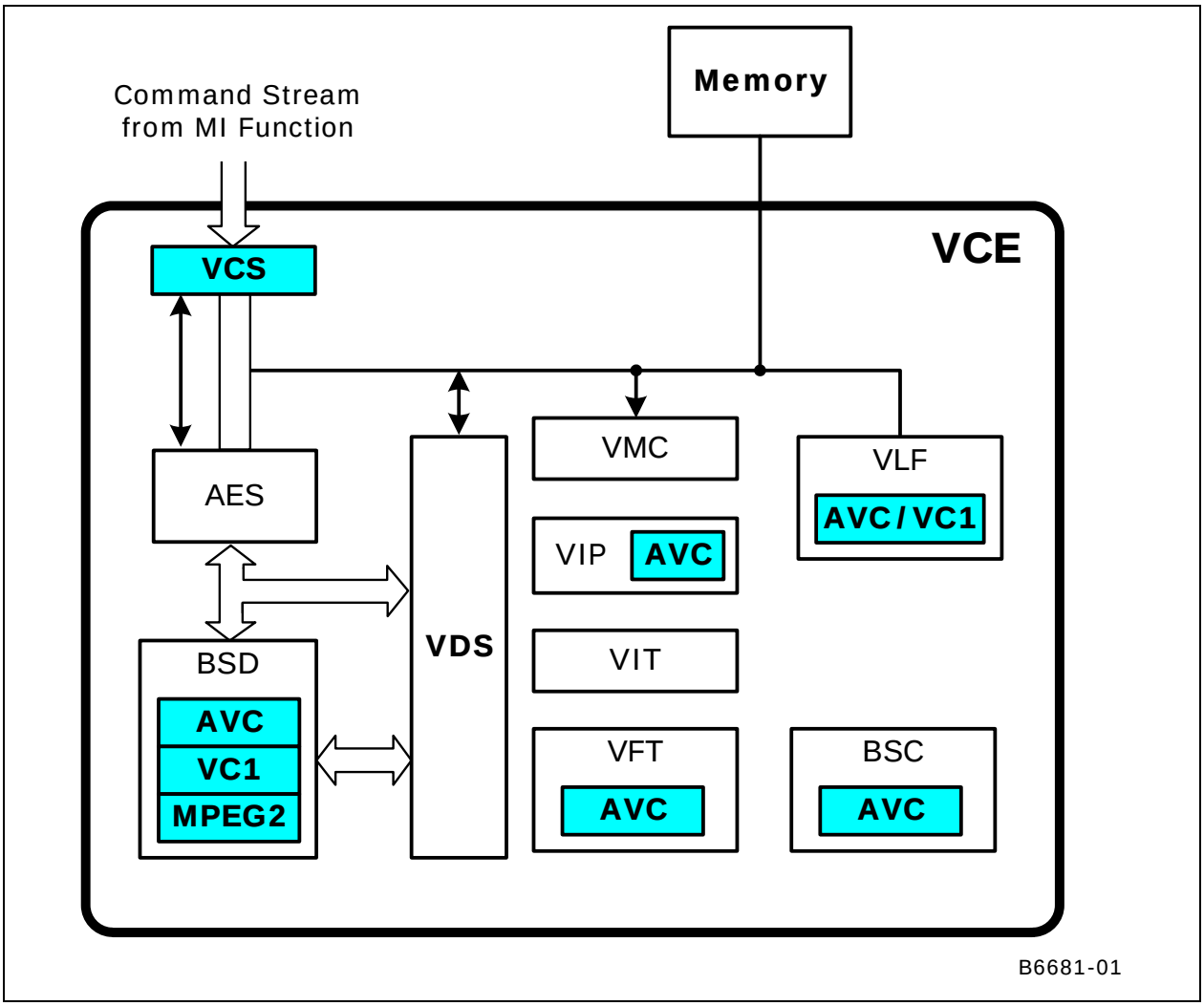
Before issuing a MEDIA_OBJECT command, all state (with the exception of 3D-pipeline-specific state) needs to be valid. Therefore the commands used to set this state need to have been issued at some point prior to the issue of MEDIA_OBJECT.

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	Interrupt table?
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	Interrupt table?
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_BYTECOUNT_SLICE	Bitstream Output Byte Count Register per Slice	RO
1240Ch	MFC_BITSTREAM_SE_BITCOUNT_SLICE	Bitstream Output Bit Count for the last Syntax Element Register	RO
12410h	MFC_AVC_CABAC_INSERTION_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

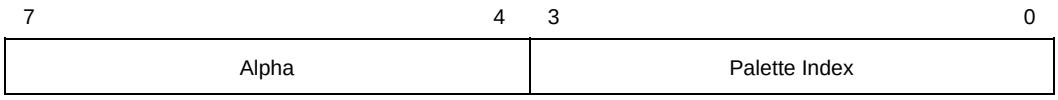
Bit	Description
T0	**Texel 0** On texture reads, this (unsigned) 1-bit value is replicated to all color channels. Format: U1
...	**...**
T7	**Texel 7** On texture reads, this (unsigned) 1-bit value is replicated to all color channels. Format: U1

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

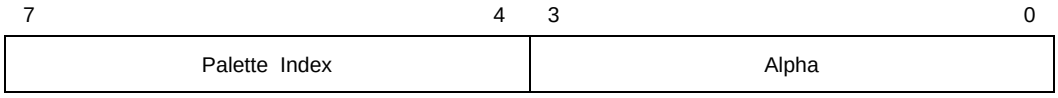
This surface format contains a 4-bit Alpha value (in the high nibble) and a 4-bit Palette Index value (in the low nibble).



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

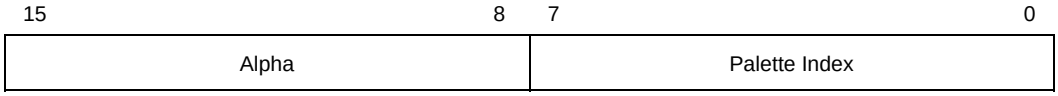
This surface format contains a 4-bit Alpha value (in the low nibble) and a 4-bit Color Index value (in the high nibble).



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

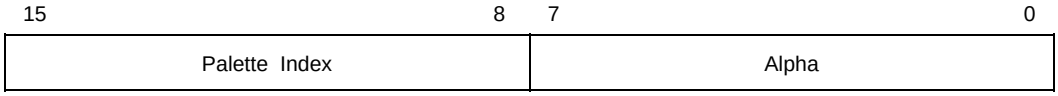
This surface format contains an 8-bit Alpha value (in the high byte) and an 8-bit Palette Index value (in the low byte).



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).



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

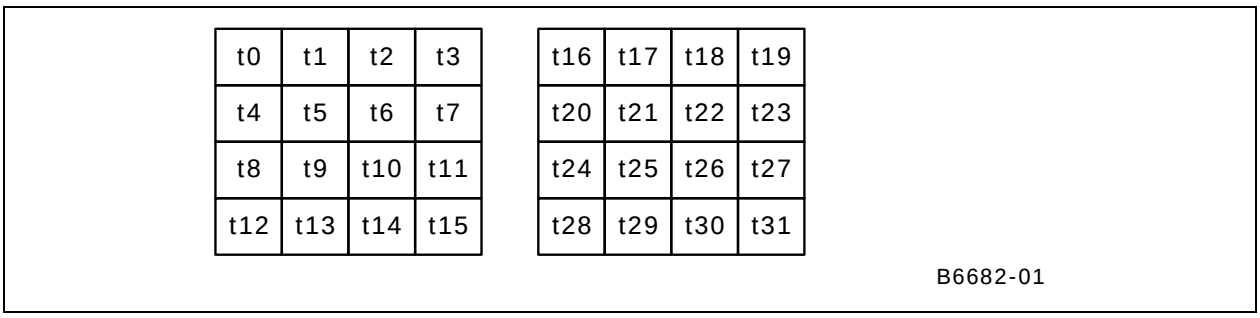
7.4 Compressed Surface Formats

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



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.

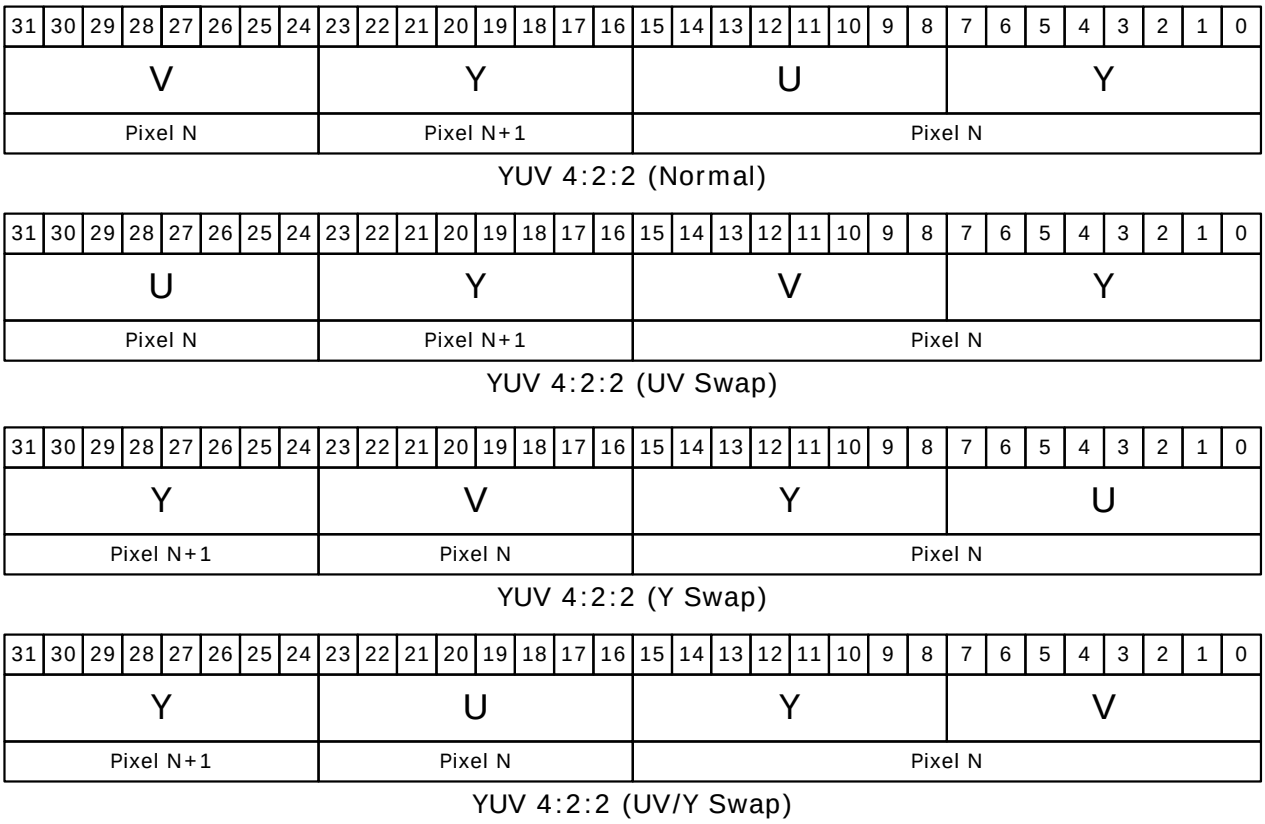
The following four YUV 4:2:2 surface formats are supported, listed with alternate names:

- 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



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.

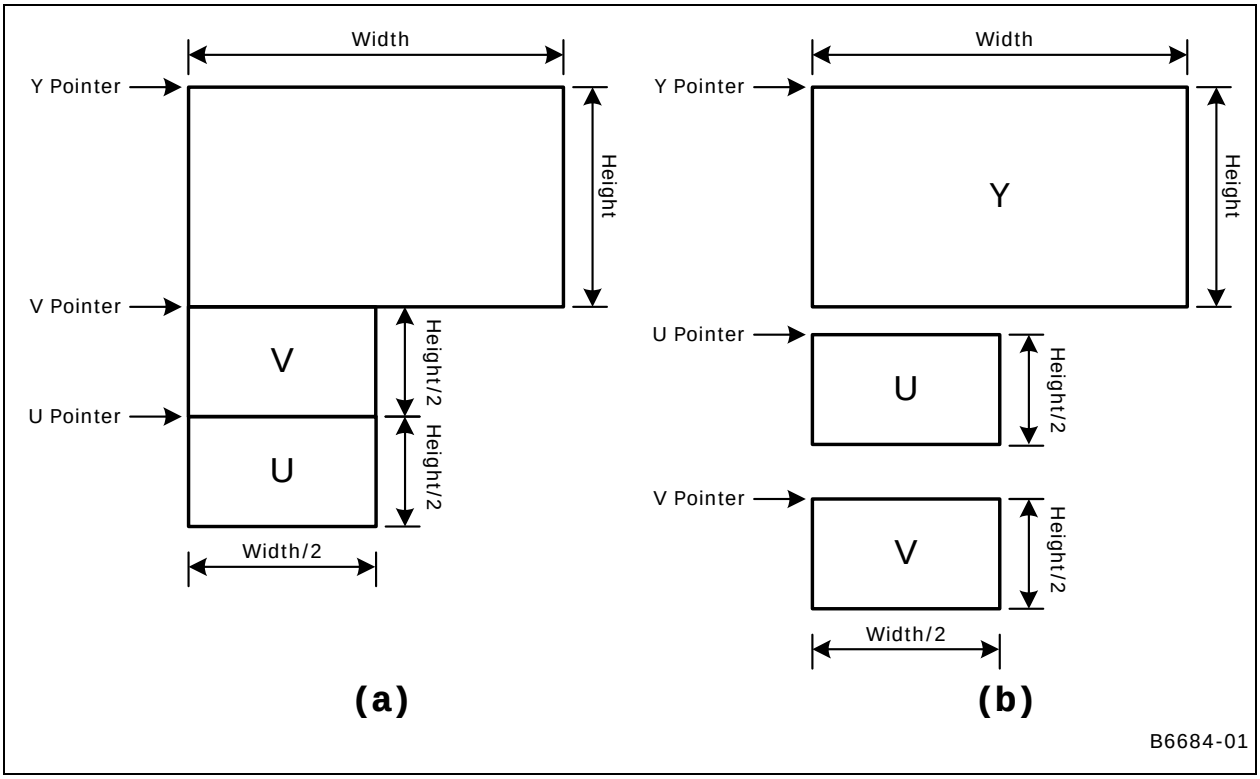
Note: There is no direct support for use of planar video surfaces as textures. The sampling engine can be used to operate on each of the 8bpp buffers separately (via a single-channel 8-bit format such as I8_UNORM). The U and V buffers can be written concurrently by using multiple render targets from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

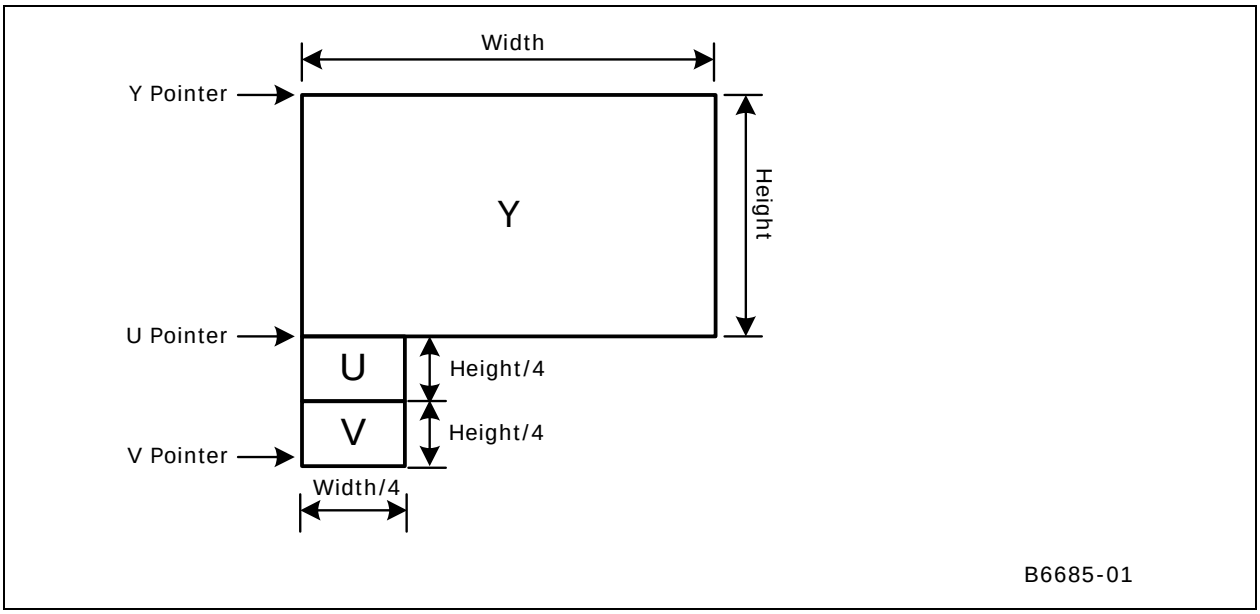
2. An alternative memory structure that the addresses of the three planes are independent but satisfy certain alignment restrictions.

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 always requires 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.

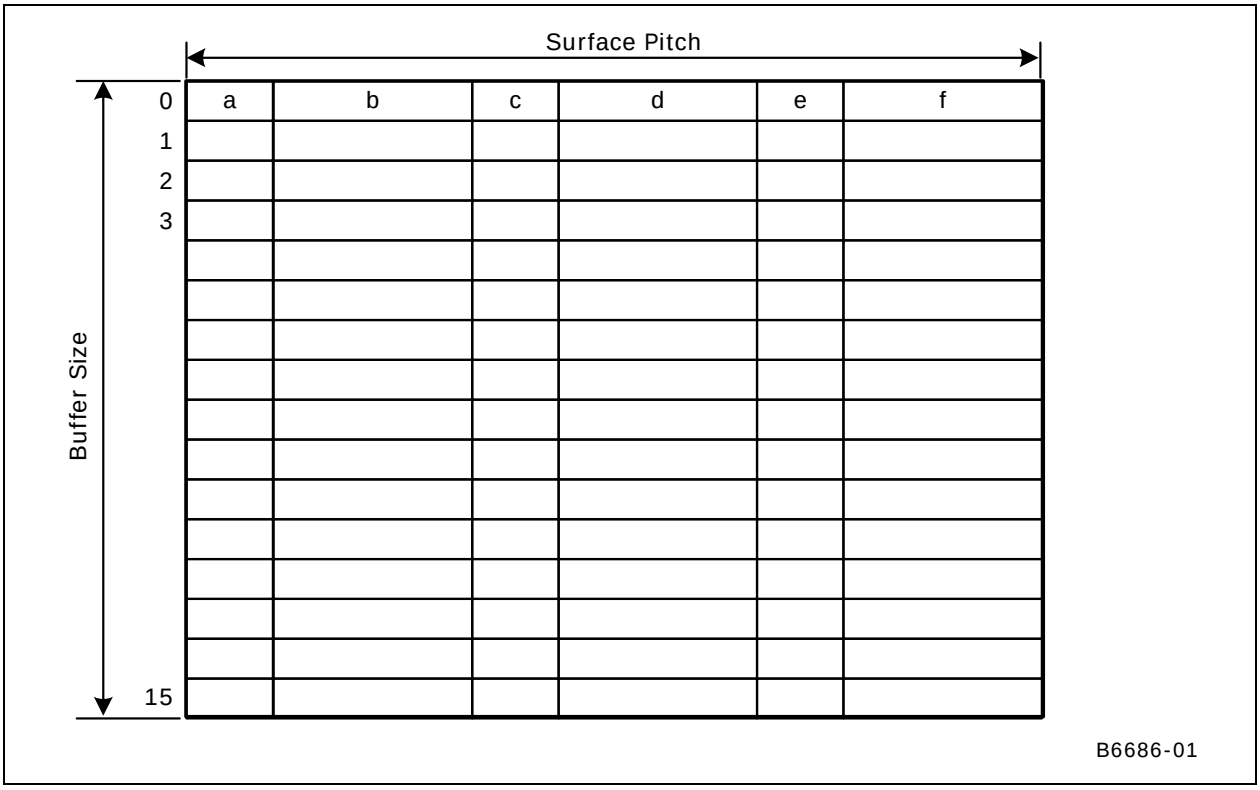
7.18 Surface Layout

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:

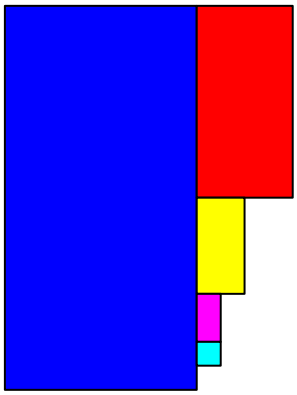
$$w_L = i * ceil\left(\frac{W_L}{i}\right)$$

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

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

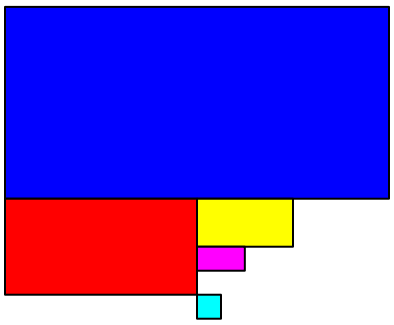
for *MIPLAYOUT_RIGHT* maps:

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



for *MIPLAYOUT_BELOW* maps:

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



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:

$$QPitch = \left(h_0 + h_1 + 11j\right)$$

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

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

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

[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.**

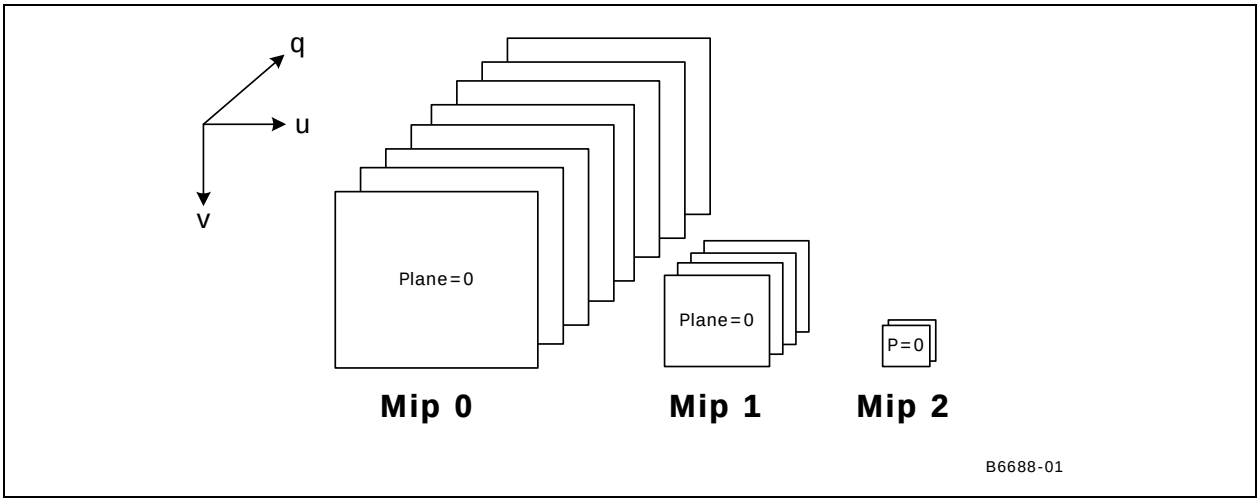
7.18.4.3 Cube Arrays [DevSNB+]

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.

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

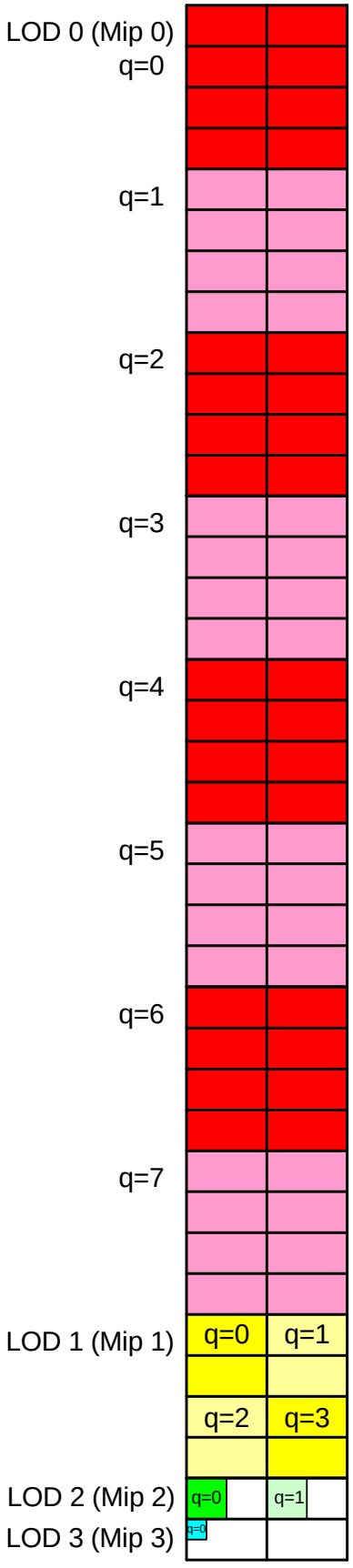
Note that it is not necessary to cache-line align in the "depth" dimension (i.e. lower case "*d*").

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

§§