G45: Volume 1a Graphics Core

Intel® 965G Express Chipset Family and Intel® G35 Express Chipset Graphics Controller

Programmer's Reference Manual (PRM)

January 2009

Revision 2.0a
Reference Number: 321391-001

Technical queries: ilg@linux.intel.com

www.intellinuxgraphics.org

You are not obligated to provide Intel with comments or suggestions regarding this document. However, should you provide Intel with comments or suggestions for the modification, correction, improvement, or enhancement of: 9a) this document; or (b) Intel products, which may embody this document, you grant to Intel a non-exclusive, irrevocable, worldwide, royalty-free license, with the right to sublicense Intel's licensees and customers, under Recipient intellectual property rights, to use and disclose such comments and suggestions in any manner Intel chooses and to display, perform, copy, make, have made, use, sell, and otherwise dispose of Intel's and its sublicensee's products embodying such comments and suggestions in any manner and via any media Intel chooses, without reference to the source.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

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

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

The Intel® 965 Express Chipset Family and Intel® G35 Express Chipset may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

I2C is a two-wire communications bus/protocol developed by Philips. SMBus is a subset of the I2C bus/protocol and was developed by Intel. Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

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

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

Contents

Figures

Tables

Revision History

Document Number	Revision Number	Description	Revision Date
24513	1.0a	Initial release.	January 2008
321391-001	2.0a	Cantiga Release	January 2009

§§

1 Introduction

This Programmer's Reference Manual (PRM) describes the architectural behavior and programming environment of the Intel® 965 Express Chipset family and Intel® G35 Express Chipset GMCH graphics devices (see Table 1-1). The GMCH's 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 term "GenX" is used throughout the PRM to refer to the Generation family of graphics devices. The devices listed in Table 1-1 are GenX devices.

Table 1-1. Supported Chipsets

Chipset Family Name	Device Name	Device Tag
Intel® Q965 Chipset Intel® Q963 Chipset Intel® G965 Chipset	82Q965 GMCH 82Q963 GMCH 82G965 GMCH	[DevBW]
Intel® G35 Chipset	82G35 GMCH	[DevBW-E]
Intel® GM965 Chipset Intel® GME965 Chipset	GM965 GMCH GME965 GMCH	[DevCL]
Intel® GM45 Chipset	GM45 GMCH GS45 GMCH GL40 GMCH	[DevCTG]
Intel® G41 Chipset Intel® G45 Chipset Intel® G43 Chipset Intel® G54 Chipset Intel® Q43 Chipset Intel® Q45 Chipset	GM41 GMCH GM43 GMCH GM54 GMCH Q43 GMCH Q45 GMCH	[DevEL]

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.

Throughout the PRM, references to "All" in a project field refters to all devices in Table 1-1.

Throughout the PRM, references to [DevBW] apply to both [DevBW] and [DevBW-E]. [DevBW-E] is referenced specifically for information that is [DevBW-E] only.

Stepping info is sometimes appended to the device tag (e.g., [DevBW-C]). Information without any device tagging is applicable to all devices/steppings.

The PRM is intended for hardware, software, and firmware designers who seek to implement or use the graphic functions of the 965 Express Chipset family, G35 Express Chipset, and the 4 Series Chipset Family. Familiarity with 2D and 3D graphics programming is assumed.

The Programmer's Reference Manual is organized into four volumes:

PRM, Volumes 1a and 1b: Graphics Core

Volume 1 covers the overall Graphics Processing Unit (GPU), without much detail on 3D, Media, or the core subsystem. Topics include the command streamer, context switching, and memory access (including tiling). The Memory Data Formats can also be found in this volume.

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

Volume 2 covers the 3D and Media pipelines in detail. This volume is where details for all of the "fixed functions" are covered, 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). It is generic threads which will be used for doing the majority of media functions. Programmable kernels will handle the algorithms for media functions such IDCT, Motion Compensation, and even Motion Estimation (used for encoding MPEG streams).

PRM, Volume 3: Display Registers

Volume 3 describes the control registers for the display. The overlay registers and VGA registers are also cover in this volume.

PRM, Volume 4: Subsystem and Cores

Volume 4 describes the GMCH programmable cores, or EUs, and the "shared functions", which are shared by more than one EU and perform functions such as I/O 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), and the Message Gateway which is 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 along with the shared functions, although the generic message send EU instruction is described with the rest of the instructions in the Instruction Set Architecture (ISA) chapters.

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

1.1 Notations and Conventions

1.1.1 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 software treats these bits as having a future, though unknown, effect. The behavior of reserved bits should be regarded as not only undefined, but unpredictable. Software should follow these guidelines in dealing with reserved bits:

- Do not depend on the states of any reserved bits when testing values of registers that contain such bits. Mask out the reserved bits before testing. Do not depend on the states of any reserved bits when storing to instruction or to a register.
- 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.

1.2 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 GenX 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.

Term	Abbr.	Definition
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.
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 GenX programs. The Data Port contains the render cache and the constant cache and performs all memory accesses requested by GenX programs except those performed by the Sampler. See DataPort.

Term	Abbr.	Definition
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 GenX SIMD instruction.
Destination Width	—	The size of each of (possibly) many elements of the destination of a GenX 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. See *Strips and Fans*.
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.
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	—	The width of each of several data elements that may be processed by a single GenX SIMD instruction.
Execution Size	ExecSize	Execution Size indicates the number of data elements processed by a GENX SIMD instruction. It is one of the GENX instruction fields and can be changed per instruction.
Execution Unit	EU	Execution Unit. An EU is a multi-threaded processor within the GENX 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 GENX 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 GenX SIMD instruction.
Extended Math Unit	EM	A Shared Function that performs more complex math operations on behalf of several EUs.

Term	Abbr.	Definition
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.
GENX Core	—	Alternative name for an EU in the GENX 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 between the vertex shader and the clipper 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. See *Geometry Shader*.
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.
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 GenX region-based GRF access.

Term	Abbr.	Definition
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.
Instruction	—	Data in memory directing an EU operation. Instructions are fetched from memory, stored in a cache and executed on one or more GenX 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 GENX ISA describes the instructions supported by a GENX 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 GenX 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 GenX instructions.
Least Significant Bit	LSB	Least Significant Bit

Term	Abbr.	Definition
MathBox	—	See Extended Math Unit
Media	—	Term for operations such as video decode and encode 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 GENX 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	Most Significant Bit
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.

Term	Abbr.	Definition
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.
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 GenX program that is supplied by the application in an high level shader language, and translated to GenX 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.

Term	Abbr.	Definition
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.
State Variable	SV	An individual state element that can be varied to change the way given primitives are rendered or media objects processed. On GenX 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.
Strips and Fans	SF	Fixed function unit whose main function is to decompose primitive topologies such as strips and fans into primitives or objects.
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 GenX 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.

Term	Abbr.	Definition
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 GenX 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 GENX 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 GENX Fixed Functions in order for a thread to return data that will be consumed either by a Fixed Function or other threads.
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	—	See URB Reference
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.

Term	Abbr.	Definition
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	—	For the most part, data is passed down the fixed function pipeline in an indirect fashion. The data is typically stored in the URB and accessed via a URB handle. When a pipeline stage passes the handle of a URB data entry to a downstream stage, it is said to make a URB reference. Note that there may be several references to the same URB data entry in the pipeline at any given time. When a downstream stage accesses the URB data entry via a URB handle, it is said to "dereference" the URB data entry. When there are no longer any references to a URB data entry within the pipeline, the URB storage can be reclaimed.
Variable Length Decode	VLD	The first stage of the video decoding pipe that consists mainly of bit-wide operations. GENX 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. Available in the VF and VS units for debugging purposes. Not unique per vertex instance.
Vertex Sequence Number	—	Unique ID for each vertex sent down the south bus that may be used to identify vertices for debugging purposes.
Vertex Shader	VS	An API-supplied program that calculates vertex attributes. Also refers to the FF unit that dispatches threads to "shade" (calculate attributes for) vertices.
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 GenX region-based GRF access.
Video Front End	VFE	The first fixed function in the GENX generic pipeline; performs fixed-function media operations.

Term	Abbr.	Definition
Viewport	VP	Post-clipped geometry is mapped to a rectangular region of the bound rendertarget(s). This rectangular region is called a viewport. Typically, the viewport is set to the full extent of the rendertarget(s), but any subregion can be used as well.
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.

§§

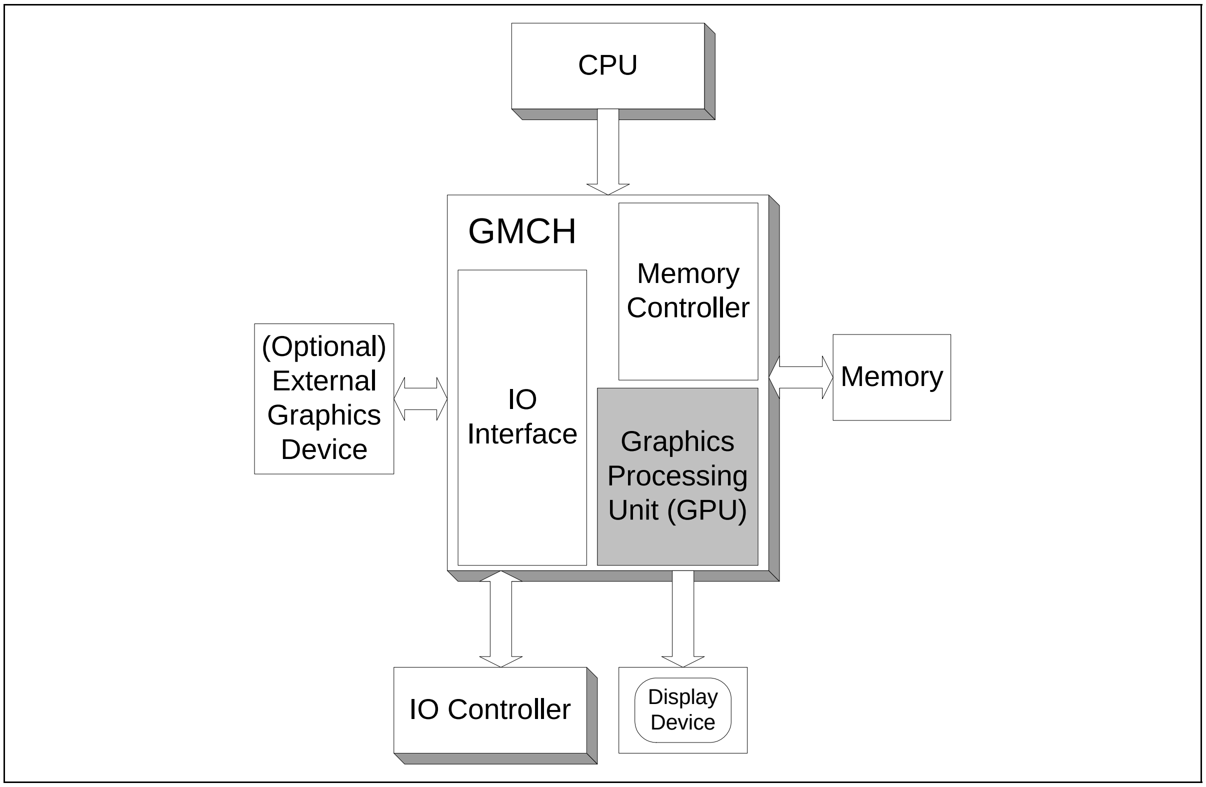
2 Graphics Device Overview

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 also to 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 can be viewed as comprising the command streamer (CS) or command parser, the Memory Interface or MI, the display interface, and (by far the largest element of the GenX family GMCH) the 3D/Media engine. This latter piece is made up of the 3D and media "fixed function" (FF) pipelines, and the GenX subsystem, which these pipelines make use of to run "shaders" and kernels.

Figure 2-1. GMCH Block Diagram

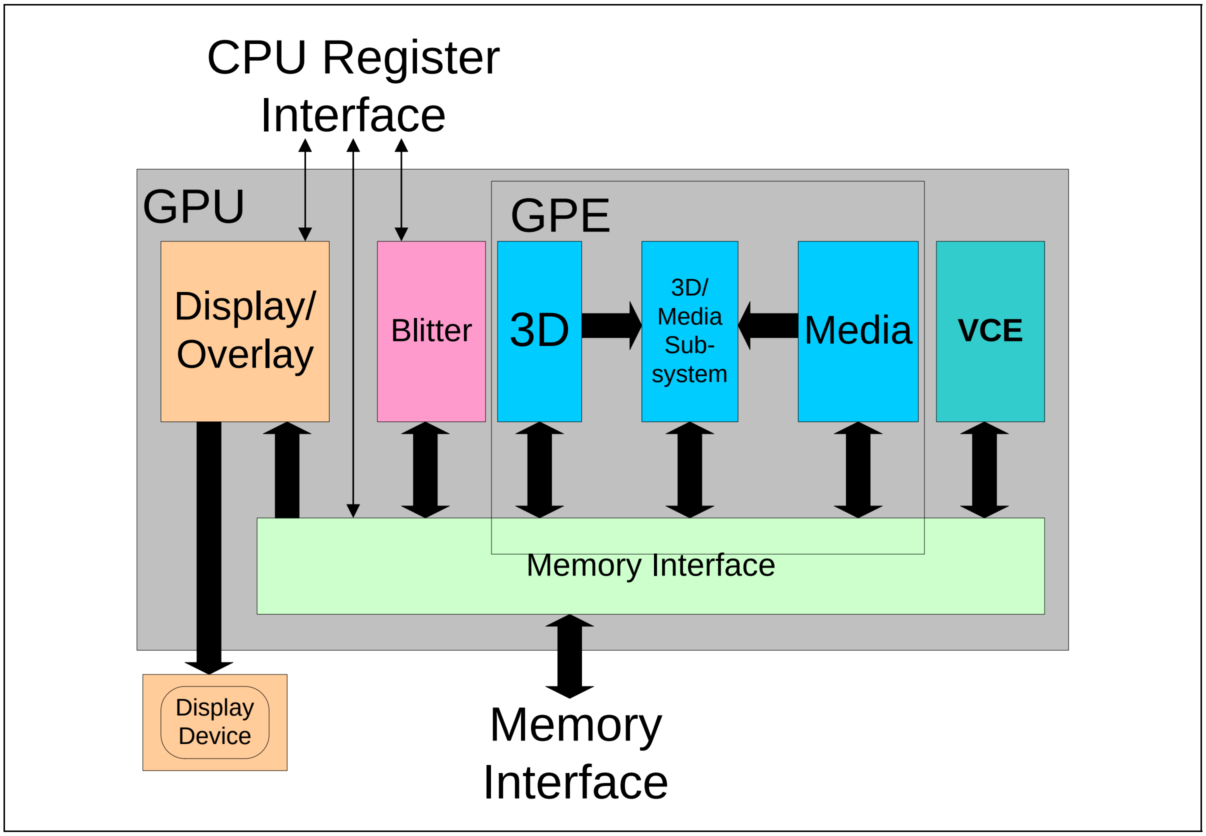


2.2 Graphics Processing Unit (GPU)

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**ransferr**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 GenX subsystem contains an array of cores, or execution units, along with a number of "shared functions", which receive and process messages at the behest 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 GENX 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 GENX GPU. It consists of the 3D and Media fixed-function pipelines, the Command Streamer (CS) unit that feeds them, and the GENX Subsystem that provides the bulk of the computations required by the pipelines.

3.2.1 Block Diagram

Figure 3-1. The Graphics Processing Engine

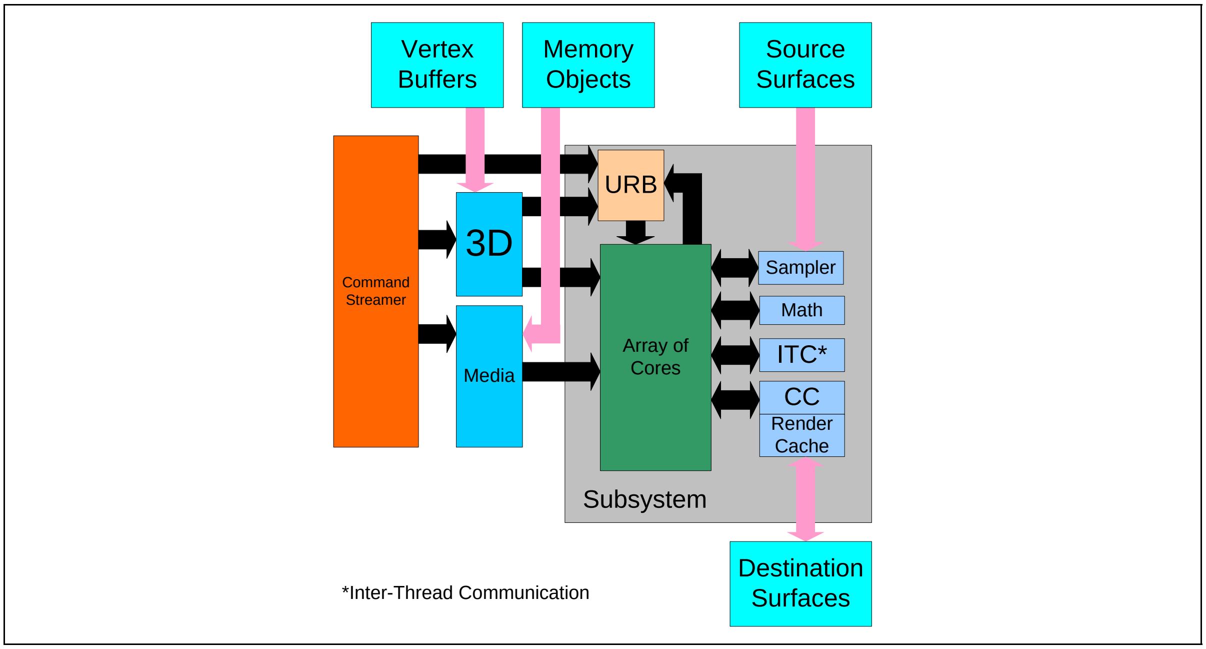
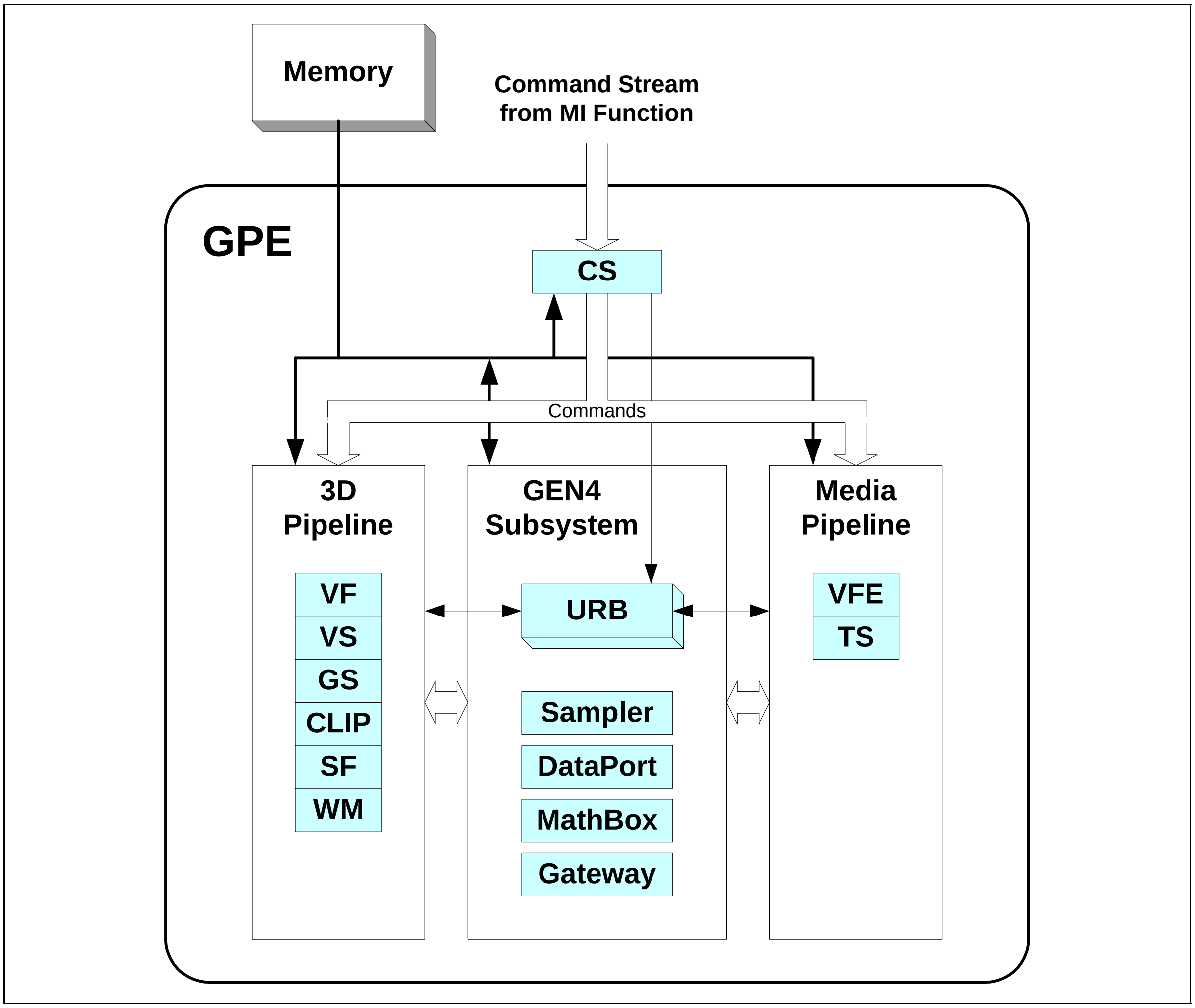


Figure 3-2. GPE Diagram Showing Fixed/Shared Functions



3.2.2 Command Stream (CS) Unit

The Command Stream (CS) unit manages the use of the 3D and Media pipelines, in that 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.

3.2.3 3D Pipeline

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

3.2.4 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 GENX threads requested by the VFE unit or as required when the pipeline is used for general processing. See *Media Pipeline Overview*.

3.2.5 GENX Subsystem

The GENX Subsystem is the collective name for the GENX 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 which manages threads running on the cores.

3.2.5.1 Execution Units (EUs)

While the number of EU cores in the GENX 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)

Debug registers (e.g., EU-enable bitmasks)

Device	# of EUs	#Threads/EU
[DevCTG] and DevEL]	10	5
All Others	8	4

3.2.6 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.

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 is not a requirement of the architecture.

Table 3-1. GenX 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	Reserved	Reserved	---
0x5	Reserved	Reserved	Reserved	---
0x6	SFID_URB	URB	Reserved	---
0x7	SFID_SPAWNER	Thread Spawner	FFID_SPAWNER	Thread Spawner
0x8	Reserved	---	FFID_VFE	Video Front End
0x9	Reserved	---	FFID_VS	Vertex Shader
0xA	Reserved	---	FFID_CS	Command Stream
0xB	Reserved	---	FFID_VF	Vertex Fetch
0xC	Reserved	---	FFID_GS	Geometry Shader
0xD	Reserved	---	FFID_CLIP	Clipper Unit
0xE	Reserved	---	FFID_SF	Strip/Fan Unit
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. This will cause each 3D FF to start a URB deallocation sequence after the current tasks are done. When the WM sees this, it will de-reference the current Constant URB Entry. Once this happens, all 3D URB entries will be deallocated (after some north bus delay). This allows the CS to set the URB fences for the media pipe. And vice versa for switching from media to 3D pipes.

Programming Restriction:

Software must ensure the current pipeline is flushed via an MI_FLUSH 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 = 0h, 26:24 = 1h, 23:16 = 04h] (Non-pipelined)
	15:1	Reserved: MBZ
	0	**Pipeline Select** 0: 3D pipeline is selected 1: Media pipeline is selected

This one bit of **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 following).

3.4.1 URB_FENCE

The URB_FENCE command is used to define the current URB allocation for those FF units that can own (write) URB entries. The FF units' allocations are specified via a set of 512-bit granular *fence pointers*, in a predefined order in the URB as shown in the diagram below. (In the discussion below, "previous" refers to the relative position in the list presented in Figure 3-3, not necessarily with respect to the order of fence pointers in the command or the order of FF units in the physical pipelines).

The URB_FENCE command is required in certain programming sequences (see programming notes below, as well as the Command Ordering Rules subsection below).

Each FF unit that can own URB entries is provided with a fence pointer that specifies the URB address immediately following that FF unit's allocated region (i.e., it identifies the end of the allocated region). The range allocated to a particular FF unit therefore starts at the previous FF unit's fence pointer and ends at its associated fence pointer. The starting fence pointer for the first (VS) fixed function is implied to be 0. URB locations starting at the fence pointer of the last FF unit in the list (CS) are effectively unusable. If a FF unit's fence pointer is identical to the previous FF unit's fence pointer, the FF unit has no URB storage allocated to it (and therefore the FF unit must either be disabled or otherwise programmed to not require its own URB entries).

The fencing and allocation of the URB is performed in a pipeline-dependent manner. The following diagrams show the layout of the URB fence regions for the 3D and Media pipelines (depending on which one is selected via PIPELINE_SELECT). In the URB_FENCE command, **Fence** values not associated with the currently selected pipeline will be ignored.

Figure 3-3. URB Allocation – 3D Pipeline

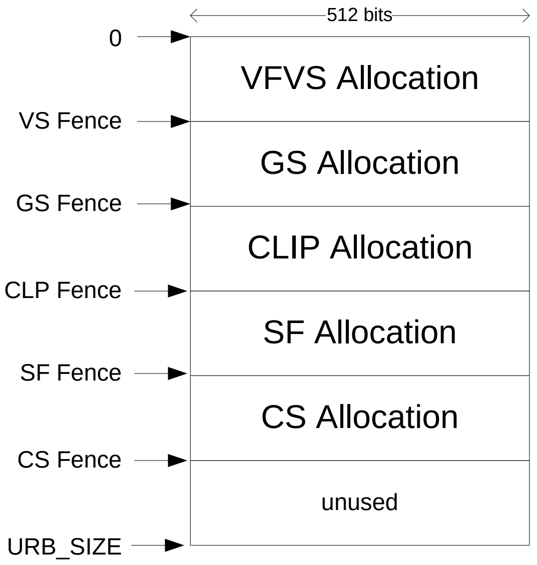
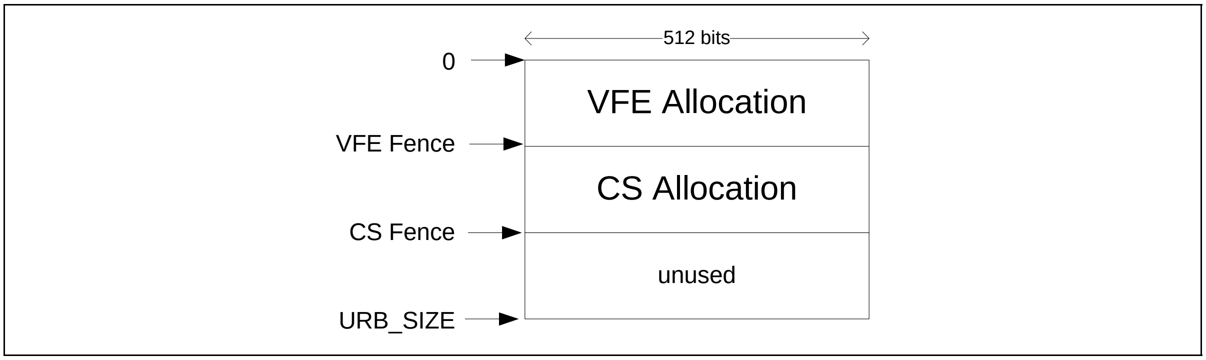


Figure 3-4 URB Allocation – Media Pipeline



Programming Notes:

1. URB Size
   a. [DevBW], [DevCL] URB_SIZE is 16KB = 256 512-bit units
   b. [DevCTG], [DevEL] URB_SIZE is 24KB = 384 512-bit units
2. On a per-fixed-function basis, software must modify (via pipeline state pointer commands) any (active) fixed-function state which relies on the size of the fixed-function's fenced URB region. If a fixed-function's URB region is repositioned within the URB, but retains the same size, the previous state is still valid. Note that changing fence pointers via URB_FENCE only affects the location of the allocated region, not the contents – i.e., no data copy is performed.
3. A URB_FENCE command must be issued subsequent to any change to the value in the GS or CLIP unit's **Maximum Number of Threads** state (via PIPELINE_STATE_POINTERS) and before any subsequent pipeline processing (e.g., via 3DPRIMITIVE or CONSTANT_BUFFER).
4. A URB_FENCE command must be issued subsequent to any change to the value in any FF unit's **Number of URB Entries** or **URB_Entry Allocation Size** state (via PIPELINE_STATE_POINTERS) and before any subsequent pipeline processing (e.g., via 3DPRIMITIVE or CONSTANT_BUFFER). Also see the Command Ordering Rules subsection below.
5. To workaround a silicon issue it is required that this instruction be programmed within a 64 byte cacheline aligned memory chunk (i.e., it must not cross a 64-byte cacheline boundary.)

URB_FENCE

<table>
<tr><td><b>Project:</b></td><td>All</td><td><b>Length Bias:</b></td><td>2</td></tr>
<tr><td colspan="4">This command is used to set the fences between URB regions owned by the fixed functions.</td></tr>
</table>

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td>
<table>
<tr><td colspan="5"><b>Command Type</b></td></tr>
<tr><td>Default Value:</td><td>3h</td><td>GFXPIPE</td><td>Format :</td><td>OpCode</td></tr>
</table>
</td></tr>
<tr><td></td><td>28:27</td><td>
<table>
<tr><td colspan="5"><b>Command SubType</b></td></tr>
<tr><td>Default Value:</td><td>0h</td><td>GFXPIPE_COMMON</td><td>Format:</td><td>OpCode</td></tr>
</table>
</td></tr>
<tr><td></td><td>26:24</td><td>
<table>
<tr><td colspan="5"><b>3D Command Opcode</b></td></tr>
<tr><td>Default Value:</td><td>0h</td><td>GFXPIPE_PIPELINED</td><td>Format:</td><td>OpCode</td></tr>
</table>
</td></tr>
<tr><td></td><td>23:16</td><td>
<table>
<tr><td colspan="5"><b>3D Command Sub Opcode</b></td></tr>
<tr><td>Default Value:</td><td>00h</td><td>URB_FENCE</td><td>Format:</td><td>OpCode</td></tr>
</table>
</td></tr>
<tr><td></td><td>15:14</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project :</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>13</td><td>
<table>
<tr><td colspan="3"><b>CS Unit URB Reallocation Request</b></td></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Format:</td><td>Enable</td><td>FormatDesc</td></tr>
<tr><td colspan="3">
<p>If set, the CS unit will perform a URB entry deallocation/reallocation action.</p>
<p><b>Note:</b> 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.</p>
<p>(The following description applies to all URB Reallocation Request bits):</p>
<p>A reallocation action is required if either (a) the region of the URB allocated to this unit changes location or size as defined by the bracketing <b>Fence</b> values, or (b) the <b>Number of URB Entries</b> or <b>URB Entry Allocation Size</b> state variables associated with this unit have been modified since the last reallocation action. Software is required to set this bit accordingly.</p>
<p>Within the context's command stream, this is the only cause of a reallocation action --- a reallocation action is <b>not</b> performed as a side effect of a change to the formentioned state variables.   Hardware will, however, take care of deallocation/reallocation resulting from context swtiches.</p>
<p>Note that all <b>Fence</b> values provided in this command (and relevant to the selected pipeline) are considered valid and provided to the active pipeline, regardless of any reallocation requests.   For example, if the 3D pipeline is selected and only the <b>CS Fence</b> is being changed, the <b>CLIP, GS, VS and SF Fence</b> values must be programmed to their correct (previous) values.</p>
</td></tr>
</table>
</td></tr>
</table>

URB_FENCE

	12	**VFE Unit URB Reallocation Request** Project: All Format: Enable FormatDesc If set, the VFE unit will perform a URB entry deallocation/reallocation action. (See **CS Unit URB Reallocation Request** description)
	11	**SF Unit URB Reallocation Request** Project: All Format: Enable FormatDesc If set, the SF unit will perform a URB entry deallocation/reallocation action. (See **CS Unit URB Reallocation Request** description)
	10	**CLIP Unit URB Reallocation Request** Project: All Format: Enable FormatDesc If set, the CLIP unit will perform a URB entry deallocation/reallocation action. (See **CS Unit URB Reallocation Request** description)
	9	**GS Unit URB Reallocation Request** Project: All Format: Enable FormatDesc If set, the GS unit will perform a URB entry deallocation/reallocation action. (See **CS Unit URB Reallocation Request** description)
	8	**VS Unit URB Reallocation Request** Project: All Format: Enable FormatDesc If set, the VS unit will perform a URB entry deallocation/reallocation action. (See **CS Unit URB Reallocation Request** description)
	7:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:30	**Reserved** Project: All Format: MBZ

URB_FENCE

	29:20	**CLIP Fence** Project: All Format: U10 representing the first 512-bit URB address beyond this unit's URB space — FormatDesc Range [DevBW], [DevCL] [GS Fence,256] [DevCTG], [DevEL] [GS Fence, 384] Indicates the URB fence value for the CLIP unit. This field is considered valid whenever the 3D pipeline is selected via PIPELINE_SELECT. Otherwise it is ignored.
	19:10	**GS Fence** Project: All Format: U10 representing the first 512-bit URB address beyond this unit's URB space — FormatDesc Range [DevBW], [DevCL] [VS Fence,256] [DevCTG], [DevEL] [VS Fence, 384] Indicates the URB fence value for the GS unit. This field is considered valid whenever the 3D pipeline is selected via PIPELINE_SELECT. Otherwise it is ignored.
	9:0	**VS Fence** Project: All Format: U10 representing the first 512-bit URB address beyond this unit's URB space — FormatDesc Range [DevBW], [DevCL] [0,256] [DevCTG], [DevEL] [0,384] Indicates the URB fence value for the VS unit. Note: When the 3D pipeline is used, the VS FF unit must be allocated URB space even if the VS function (i.e., "vertex shading") is disabled. The VF unit utilizes Vertex URB Entries (VUEs) allocated to the VS in order to input vertex data to the 3D pipeline even if vertex shading is not enabled. This field is considered valid whenever the 3D pipeline is selected via PIPELINE_SELECT. Otherwise it is ignored.
2	31	**Reserved** Project: All Format: MBZ

URB_FENCE

30:20	**CS Fence**		
	Project:	All	
	Format:	U11 representing the first 512-bit URB address beyond this unit's URB space	FormatDesc
	Range	[VFE Fence,256] (Media) or [SF Fence,256] (3D Pipe) [DevBW], [DevCL] [VFE Fence,384] (Media) or [SF Fence,384] (3D Pipe) [DevCTG], [DevEL]	
	Indicates the URB fence value for the CS unit. This field is always considered valid, as it is relevant regardless of the currently selected pipeline.		
19:10	**VFE Fence**		
	Project:	All	
	Format:	U10 representing the first 512-bit URB address beyond this unit's URB space	FormatDesc
	Range	[DevBW], [DevCL] [0,256] [DevCTG], [DevEL] [0,384]	
	Indicates the URB fence value for the VFE unit. This field is considered valid whenever the Media pipeline is selected via PIPELINE_SELECT. Otherwise it is ignored.		
9:0	**SF Fence**		
	Project:	All	
	Format:	U10 representing the first 512-bit URB address beyond this unit's URB space	FormatDesc
	Range	[DevBW], [DevCL] [CLIP Fence,256] [DevCTG], [DevEL] [CLIP Fence,384]	
	Indicates the URB fence value for the SF unit. This field is considered valid whenever the 3D pipeline is selected via PIPELINE_SELECT. Otherwise it is ignored.		

3.5 Constant URB Entries (CURBEs)

3.5.1 Overview

It is anticipated that threads will need to access some amount of non-immediate constant data, e.g., a matrix from a VS kernel. While the DataPort can be used to read ("pull") this data from a memory buffer, doing so may incur a performance penalty due to the latency of the access. In order to provide a higher-performance path, both pipelines are provided with the ability to preload ("push") data from a memory buffer into the URB and have portions of that data automatically included in subsequent thread payloads. These pushed constants will then be immediately available for use by the thread (at the expense of increased GRF allocation, dispatch latency, etc.).

The mechanism to push constants into thread payloads is the *Constant URB Entry* (CURBE). The CURBE is a special URB entry (owned by the CS unit) used to store the constant data. Software can issue the CONSTANT_BUFFER command to specify the source Constant Buffer in memory. Upon receipt of that command, the CS unit will read the Constant Buffer data from memory and write the data into the CURBE. Fixed functions of the pipeline can be programmed to include their subset of the CURBE data in thread payloads.

3.5.2 Multiple CURBE Allocation

There is only one "current" CURBE state provided by the architecture. Portions of the current CURBE is available to the various fixed-function stages of the pipelines. However, in order to avoid having to flush the pipeline prior to modifying the contents of the current CURBE, the GPE is supplied with the ability to pipeline changes to the current CURBE. This support comes in the form of a set of CURBEs that can be maintained in the URB. A region of the URB can be allocated to the CS unit (see URB_FENCE command) to hold this set of CURBEs. Within that region, software can define a set of up to 4 *Constant URB Entries* (CURBEs) – (see CS_URB_STATE command).

When a CONSTANT_BUFFER command is received, an attempt is made to find an unused CURBE within the set. If one is found, it is used as the destination of the memory read, and the handle of that CURBE is passed down the pipeline without incurring a pipeline flush performance penalty. Fixed functions will switch to using the new CURBE as the handle travels down the pipeline. When the handle reaches the end of the pipeline, the previous CURBE is marked as unused.

If a CONSTANT_BUFFER command is encountered and there is only one CURBE allocated and it is in use, the CS unit will implicitly wait for the pipeline to drain and the CURBE to become available to be overwritten. Due to the performance impact of modifying the CURBE when only a single CURBE is allocated, it is recommended that software operate with a single CURBE allocation only if (a) the CURBE is large enough to make multiple allocations undesirable, and/or (b) it is anticipated that the constant data will remain static for long processing periods (thus amortizing the impact of modifying it).

3.5.3 CS_URB_STATE

<table>
<tr><th colspan="3">CS_URB_STATE</th></tr>
<tr><td colspan="2"><b>Project:</b> All</td><td><b>Length Bias:</b> 2</td></tr>
<tr><td colspan="3">The CS_URB_STATE packet is used to define the number and size of CURBEs contained within the CS unit’s allocated URB region.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><table><tr><td colspan="5"><b>Command Type</b></td></tr><tr><td>Default Value:</td><td>3h</td><td>GFXPIPE</td><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><table><tr><td colspan="5"><b>Command SubType</b></td></tr><tr><td>Default Value:</td><td>0h</td><td>GFXPIPE_COMMON</td><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>26:24</td><td><table><tr><td colspan="5"><b>3D Command Opcode</b></td></tr><tr><td>Default Value:</td><td>0h</td><td>GFXPIPE_PIPELINED</td><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>23:16</td><td><table><tr><td colspan="5"><b>3D Command Sub Opcode</b></td></tr><tr><td>Default Value:</td><td>01h</td><td>CS_URB_STATE</td><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>15:8</td><td><table><tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><table><tr><td colspan="3"><b>DWord Length</b></td></tr><tr><td>Default Value:</td><td>0h</td><td>Excludes DWord (0,1)</td></tr><tr><td>Format:</td><td>=n</td><td>Total Length - 2</td></tr><tr><td>Project:</td><td colspan="2">All</td></tr></table></td></tr>
<tr><td>1</td><td>31:9</td><td><table><tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>8:4</td><td><table><tr><td colspan="3"><b>URB Entry Allocation Size</b></td></tr><tr><td>Project:</td><td colspan="2">All</td></tr><tr><td>Format:</td><td>U5 count (of 512-bit units) – 1</td><td>FormatDesc</td></tr><tr><td>Range</td><td colspan="2">[0,31] = [1,32] 512-bit units = [2,64] 256-bit URB rows</td></tr><tr><td colspan="3">Specifies the length of each URB entry owned by the CS unit.</td></tr></table></td></tr>
<tr><td></td><td>3</td><td><table><tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">CS_URB_STATE</th></tr>
<tr><td></td><td>2:0</td><td>
<table>
<tr><th colspan="3">Number of URB Entries</th></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Format:</td><td>U3 count of entries</td><td>FormatDesc</td></tr>
<tr><td>Range</td><td colspan="2">[0,4]</td></tr>
<tr><td colspan="3">Specifies the number of URB entries that are used by the CS unit.</td></tr>
</table>
</td></tr>
</table>

3.5.4 CONSTANT_BUFFER

CONSTANT_BUFFER

Project:	All	Length Bias:	2

The CONSTANT_BUFFER packet is used to define the memory address of data that will be read by the CS unit and stored into the current CURBE entry.

Programming Notes:

- Issuing a CONSTANT_BUFFER packet with **Valid** set when the CS unit does not have any CURBE entries allocated in the URB results in UNDEFINED behavior.
- 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.

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: 02h CONSTANT_BUFFER Format: OpCode
	15:9	**Reserved** Project: All Format: MBZ
	8	**Valid** Project: All Format: Enable FormatDesc If TRUE, a Constant Buffer will be defined and possibly used in the pipeline (depending on FF unit state programming). The **Buffer Starting Address** and **Buffer Length** fields are valid. If FALSE, the Constant Buffer becomes undefined and unused. The **Buffer Starting Address** and **Buffer Length** fields are ignored. The FF unit state descriptors must not specify the use of CURBE data, or behavior is UNDEFINED.

CONSTANT_BUFFER

	7:0	**DWord Length** Default Value: | 0h | Excludes DWord (0,1) Format: | =n | Total Length - 2 Project: | All
1	31:6	**Buffer Starting Address** Project: | All Format: | GeneralStateOffset[31:6] or GraphicsAddress[31:6] (see below) | FormatDesc If **Valid** is set and INSTPM<**CONSTANT_BUFFER Address Offset Disable**> is clear (enabled), this field defines the location of the memory-resident constant data via a 64Byte-granular offset from the **General State Base Address**. If **Valid** is set and INSTPM<**CONSTANT_BUFFER Address Offset Disable**> is set (disabled), this field defines the location of the memory-resident constant data via a 64Byte-granular Graphics Address (not offset). **Programming Notes** Constant Buffers can only be allocated in linear (not tiled) graphics memory Constant Buffers can only be mapped to Main Memory (UC)
	5:0	**Buffer Length** Project: | All Format: | U6 Count-1 in 512-bit units | FormatDesc If **Valid** is set, this field specifies the length of the constant data to be loaded from memory into the CURBE in 512-bit units (minus one). The length must be less than or equal to the **URB Entry Allocation Size** specified via the CS_URB_STATE command.

3.6 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 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	CS unit reads from **CURBE Constant Buffers** via CONSTANT_BUFFER when INSTPM< **CONSTANT_BUFFER Address Offset Disable**> is clear (enabled).
	3D Pipeline FF state read by the 3D FF units, as referenced by state pointers passed via 3DSTATE_PIPELINE_POINTERS.
	Media pipeline FF state, as referenced by state pointers passed via MEDIA_PIPELINE_POINTERS.
	DataPort memory accesses resulting from **'stateless' DataPort Read/Write requests**. See *DataPort* for a definition of the 'stateless' form of requests.
General State Base Address	Sampler reads of **Sampler State** data and associated **Default Color State** data
	Viewport states used by CLIP, SF, and WM/CC
	COLOR_CALC_STATE
General State 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** data, 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 unit reads from **CURBE Constant Buffers** via CONSTANT_BUFFER when INSTPM< **CONSTANT_BUFFER Address Offset Disable**> is set (disabled).
	CS writes resulting from 3D_CONTROL
	All VF unit memory accesses (**Index Buffers**, **Vertex Buffers**)
	All Sampler **Surface Memory Data** accesses (texture fetch, etc.)
	All **DataPort memory accesses** except 'stateless' DataPort Read/Write requests (e.g., RT accesses.) See *DataPort* 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)

The following notation is used in the PRM 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)
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)

3.6.1 STATE_BASE_ADDRESS

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

STATE_BASE_ADDRESS

Project:	All	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_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.
- MI_FLUSH command with ISC invalidate bit set should always be programmed prior to STATE_BASE_ADDRESS command.

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: 4h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All

STATE_BASE_ADDRESS

<table>
<tr><td>1</td><td>31:12</td><td>
<table>
<tr><th colspan="3">General State Base Address</th></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Format:</td><td>GraphicsAddress[31:12]</td><td>FormatDesc</td></tr>
<tr><td colspan="3">Specifies the 4K-byte aligned base address for general state accesses. See Table 3-2 for details on where this base address is used.</td></tr>
</table>
</td></tr>
<tr><td></td><td>11:1</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project :</td><td>All</td><td>Format :</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>0</td><td>
<table>
<tr><th colspan="3">Modify Enable</th></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Format:</td><td>Enable</td><td>FormatDesc</td></tr>
<tr><td colspan="3">The address in this dword is updated only when this bit is set.</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>Disable</td><td>Ignore the updated address</td><td>All</td></tr>
<tr><td>1h</td><td>Enable</td><td>Modify the address</td><td>All</td></tr>
</table>
</td></tr>
<tr><td>2</td><td>31:12</td><td>
<table>
<tr><th colspan="3">Surface State Base Address</th></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Format:</td><td>GraphicsAddress[31:12]</td><td>FormatDesc</td></tr>
<tr><td colspan="3">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.</td></tr>
</table>
</td></tr>
<tr><td></td><td>11:1</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>0</td><td>
<table>
<tr><th colspan="3">Modify Enable</th></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Format:</td><td>Enable</td><td>FormatDesc</td></tr>
<tr><td colspan="3">The address in this dword is updated only when this bit is set.</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>Disable</td><td>Ignore the updated address</td><td>All</td></tr>
<tr><td>1h</td><td>Enable</td><td>Modify the address</td><td>All</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th colspan="3">STATE_BASE_ADDRESS</th></tr>
<tr><td>3</td><td>31:12</td><td>
<table>
<tr><th colspan="3">Indirect Object Base Address</th></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Format:</td><td>GraphicsAddress[31:12]</td><td>FormatDesc</td></tr>
<tr><td colspan="3">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.</td></tr>
</table>
</td></tr>
<tr><td></td><td>11:1</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>0</td><td>
<table>
<tr><th colspan="3">Modify Enable</th></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Format:</td><td>Enable</td><td>FormatDesc</td></tr>
<tr><td colspan="3">The address in this dword is updated only when this bit is set.</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>Disable</td><td>Ignore the updated address</td><td>All</td></tr>
<tr><td>1h</td><td>Enable</td><td>Modify the address</td><td>All</td></tr>
</table>
</td></tr>
<tr><td>4</td><td>31:12</td><td>
<table>
<tr><th colspan="3">General State Access Upper Bound</th></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Format:</td><td>GraphicsAddress[31:12]</td><td>FormatDesc</td></tr>
<tr><td colspan="3">Specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address for general state accesses. This includes all accesses that are offset from <b>General State Base Address</b> (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 <b>General State Base Address</b>.</td></tr>
</table>
</td></tr>
<tr><td></td><td>11:1</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>0</td><td>
<table>
<tr><th colspan="3">Modify Enable</th></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Format:</td><td>Enable</td><td>FormatDesc</td></tr>
<tr><td colspan="3">The bound in this dword is updated only when this bit is set.</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>Disable</td><td>Ignore the updated bound</td><td>All</td></tr>
<tr><td>1h</td><td>Enable</td><td>Modify the bound</td><td>All</td></tr>
</table>
</td></tr>
</table>

STATE_BASE_ADDRESS

5	31:12	**Indirect Object Access Upper Bound** Project: All Format: GraphicsAddress[31:12] FormatDesc 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
	0	**Modify Enable** Project: All Format: Enable FormatDesc 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

3.7 State Invalidation ([DevCTG+])

The STATE_POINTER_INVALIDATE command is provided as an optional mechanism to invalidate 3D/Media state pointers and pointers to constant data. This is sometimes desirable to prevent prefetching of state between the time the pointed-to state is no longer needed, and the time the commands above are re-issued to point to new state.

3.7.1 STATE_POINTER_INVALIDATE ([DevCTG+])

STATE_POINTER_INVALIDATE

Project:	[DevCTG]	**Length Bias:**	1

The STATE_POINTER_INVALIDATE command marks the state pointers of the selected type(s) as invalid. The corresponding state pointer command must be issued again prior to attempting any rendering operations that depend on the state whose pointers have been marked as invalid.

The pointers initialized by the following commands are (potentially) invalidated by this command:

- 3DSTATE_PIPELINE_POINTERS
- 3DSTATE_CC_POINTERS
- CONSTANT_BUFFER
- MEDIA_STATE_POINTERS

DWord	Bit	Description
0	31:29	**Command Type** Default Value: GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: GFXPIPE_SINGLE_DW Format: OpCode
	26:24	**3D Command Opcode** Default Value: GFXPIPE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: STATE_POINTER_INVALIDATE Format: OpCode
	15:3	**Reserved** Project: All Format: MBZ
	2	**Pipelined State Pointers Invalidate** Project: All Format: Invalidate Enable The pointers initialized with the last 3DSTATE_PIPELINED_POINTERS are marked as invalid if this bit is set. Said pointers are unaffected if this bit is clear.

STATE_POINTER_INVALIDATE

	1	**Constant Buffer Invalidate** Project: All Format: Invalidate Enable The pointer initialized with the last CONSTANT_BUFFER is marked as invalid. Said pointer is unaffected if this bit is clear.
	0	**Media State Pointers Invalidate** Project: All Format: Invalidate Enable The pointers initialized with the last MEDIA_STATE_POINTERS are marked as invalid. Said pointers are unaffected if this bit is clear.

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

STATE_PREFETCH

	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] | FormatDesc 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) | FormatDesc 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

<table>
<tr><td colspan="4"><b>STATE_SIP</b></td></tr>
<tr><td><b>Project:</b></td><td>All</td><td><b>Length Bias:</b></td><td>2</td></tr>
<tr><td colspan="4">The STATE_SIP command specifies the starting instruction location of the System Routine that is shared by all threads in execution.</td></tr>
</table>

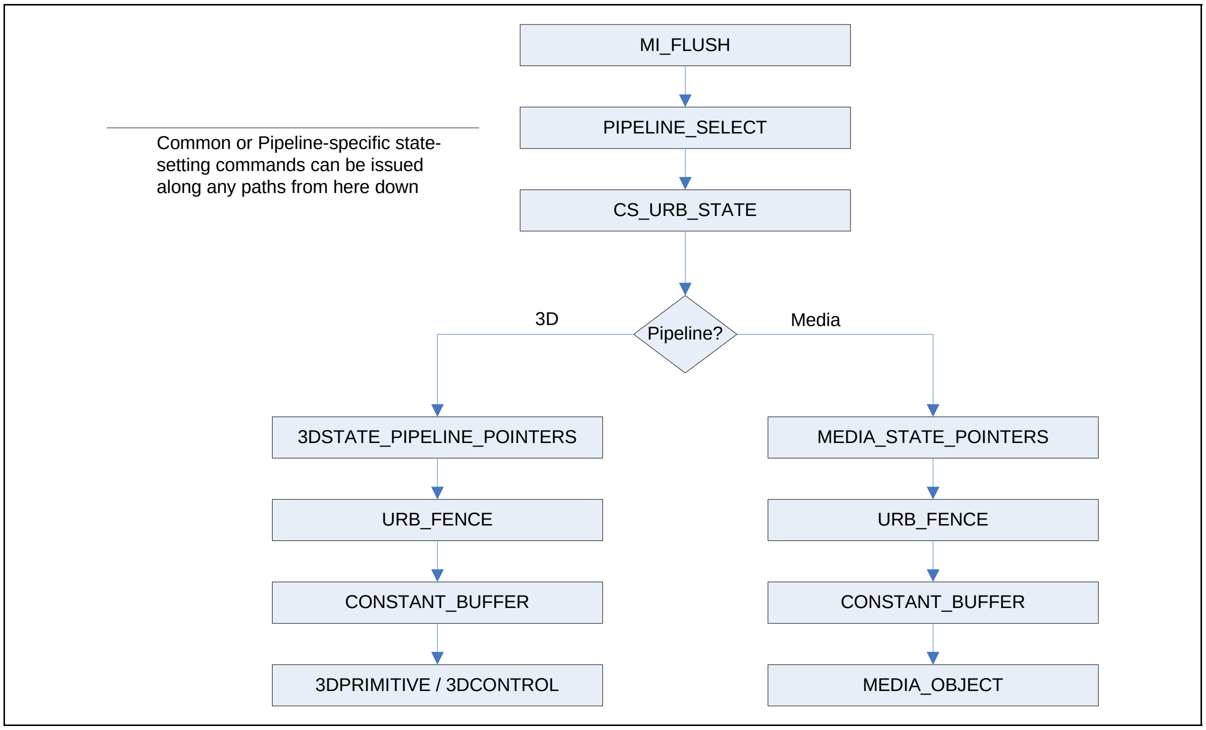
<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><table>
<tr><td colspan="5"><b>Command Type</b></td></tr>
<tr><td>Default Value:</td><td>3h</td><td>GFXPIPE</td><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>28:27</td><td><table>
<tr><td colspan="5"><b>Command SubType</b></td></tr>
<tr><td>Default Value:</td><td>0h</td><td>GFXPIPE_COMMON</td><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>26:24</td><td><table>
<tr><td colspan="5"><b>3D Command Opcode</b></td></tr>
<tr><td>Default Value:</td><td>1h</td><td>GFXPIPE_NONPIPELINED</td><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>23:16</td><td><table>
<tr><td colspan="5"><b>3D Command Sub Opcode</b></td></tr>
<tr><td>Default Value:</td><td>02h</td><td>STATE_SIP</td><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>15:8</td><td><table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>7:0</td><td><table>
<tr><td colspan="3"><b>DWord Length</b></td></tr>
<tr><td>Default Value:</td><td>0h</td><td>Excludes DWord (0,1)</td></tr>
<tr><td>Format:</td><td>=n</td><td>Total Length - 2</td></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
</table></td></tr>
<tr><td>1</td><td>31:4</td><td><table>
<tr><td colspan="3"><b>System Instruction Pointer (SIP)</b></td></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Format:</td><td>GeneralStateOffset[31:4]</td><td>FormatDesc</td></tr>
<tr><td colspan="3">Specifies the instruction address of the system routine associated with the current context as a 128-bit granular offset from the <b>General State Base Address</b>. SIP is shared by all threads in execution. The address specifies the double quadword aligned instruction location.</td></tr>
</table>
<table>
<tr><th>Errata</th><th>Description</th><th>Project</th></tr>
<tr><td><b>BWT007</b></td><td>Instructions pointed at by offsets from General State Base must be contained within 32-bit physical address space (that is, must map to memory pages under 4G.)</td><td><b>[DevBW-A]</b></td></tr>
</table></td></tr>
</table>

STATE_SIP					
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.



Common or Pipeline-specific state-setting commands can be issued along any paths from here down

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 3.3 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 sub-section 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 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.

3.10.10 MEDIA_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 Graphics Command Formats

4.1 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 GenX 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 (includes Trusted Ops)
2	2D Rendering (xxx_BLT_xxx)
3	Graphics Pipeline (3D and Media)
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.

4.1.1 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.

4.1.2 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.

4.1.3 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.

4.1.4 Video Codec Commands

4.1.4.1 Command Header

The Command Headers are shown in the following tables.

Table 4-1. 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 Reserved	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

Bits						
TYPE	31:29	28:24		23	22	21:0
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 4-2. 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	0X	XXX	XX			
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	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
Reserved (for VC1 Enc)	011	10	010	010		subopcode	DWord Count
Reserved	011	10	010	011-111		subopcode	DWord Count

Bits							
TYPE	31:29	28:24		23	22	21:0	
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			

4.2 Command Map

This section provides a map of the graphics command opcodes.

4.2.1 Memory Interface Command Map

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

Table 4-3. Memory Interface Commands for RCP

Opcode (28:23)	Command	Comments
1-DWord		
00h	MI_NOOP	
01h	Reserved	
02h	MI_USER_INTERRUPT	
03h	MI_WAIT_FOR_EVENT	
04h	MI_FLUSH	
05h	MI_ARB_CHECK	
06h	Reserved	
07h	MI_REPORT_HEAD	
08h	MI_ARB_ON_OFF	[DevCTG], [DevEL]
09h	Reserved	
0Ah	MI_BATCH_BUFFER_END	
0Bh–0Fh	Reserved	
2+ DWord		
10h	Reserved	
11h	MI_OVERLAY_FLIP	[pre-DevCTG]
12h	MI_LOAD_SCAN_LINES_INCL	
13h	MI_LOAD_SCAN_LINES_EXCL	

Opcode (28:23)	Command	Comments
14h	MI_DISPLAY_BUFFER_INFO [DevBW], [DevCL] MI_DISPLAY_FLIP [DevCTG], [DevEL]	
15h	Reserved	
16h	MI_SEMAPHORE_MBOX	[DevCTG], [DevEL]
17h	Reserved	
18h	MI_SET_CONTEXT	
19h–1Fh	Reserved	
Store Data		
20h	MI_STORE_DATA_IMM	
21h	MI_STORE_DATA_INDEX	
22h	MI_LOAD_REGISTER_IMM	
23h	MI_UPDATE_GTT	[DevCTG], [DevEL]
24h	MI_STORE_REGISTER_MEM	
25h	MI_PROBE	
26h	Reserved	
27h–2Fh	Reserved	
Ring/Batch Buffer		
30h	Reserved	
31h	MI_BATCH_BUFFER_START	
32h–3Fh	Reserved	

Table 4-4. Memory Interface Commands for VCCP

Opcode (28:23)	Command	Comments
1-DWord		
00h	MI_NOOP	
01h	Reserved	
02h	MI_USER_INTERRUPT	
03h	Reserved	
04h	MI_FLUSH	
05h	MI_ARB_CHECK	
06-09h	Reserved	
0Ah	MI_BATCH_BUFFER_END	
0Bh–0Fh	Reserved	
2- DWord		
10h–1Fh	Reserved	
Store Data		
20h	MI_STORE_DATA_IMM	
21h	MI_STORE_DATA_INDEX	
22h–2Fh	Reserved	
Ring/Batch Buffer		
30h	Reserved	
31h	MI_BATCH_BUFFER_START	
32h–3Fh	Reserved	

4.2.2 2D Command Map

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

Opcode (28:22)	Command	Comments
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	
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	

4.2.3 3D/Media Command Map

Pipeline Type (28:27)	Opcode	Sub Opcode	Command	Definition Chapter
Common (pipelined)	**Bits 26:24**	**Bits 23:16**		
0h	0h	00h	URB_FENCE	Graphics Processing Engine
0h	0h	01h	CS_URB_STATE	Graphics Processing Engine
0h	0h	02h	CONSTANT_BUFFER	Graphics Processing Engine
0h	0h	03h	STATE_PREFETCH	Graphics Processing Engine
0h	0h	04h-FFh	Reserved	
Common (non-pipelined)	**Bits 26:24**	**Bits 23:16**		
0h	1h	00h	Reserved	n/a
0h	1h	01h	STATE_BASE_ADDRESS	Graphics Processing Engine
0h	1h	02h	STATE_SIP	Graphics Processing Engine
0h	1h	03h–FFh	Reserved	n/a
Reserved	**Bits 26:24**	**Bits 23:16**		
0h	2h–7h	XX	Reserved	n/a
Pipeline Type (28:27)	**Opcode**	**Sub Opcode**	**Command**	**Definition Chapter**
Single DW	**Opcode (26:24)**	**Bits 23:16**		
1h	0h	00h-01h	Reserved	n/a
1h	0h	02h	STATE_POINTER_INVALIDATE [DevCTG], [DevEL]	Graphics Processing Engine
1h	0h	03h-0Ah	Reserved	n/a
1h	0h	0Bh	3DSTATE_VF_STATISTICS	Vertex Fetch
1h	0h	0Ch-FFh	Reserved	n/a
1h	1h	00h-03h	Reserved	n/a
1h	1h	04h	PIPELINE_SELECT	Graphics Processing Engine
1h	1h	05h-FFh	Reserved	n/a
1h	2h-7h	XX	Reserved	n/a

Media	**Opcode (26:24)**	**Bits 23:16**		
2h	0h	00h	MEDIA_STATE_POINTERS	Media
2h	1h	00h	MEDIA_OBJECT	Media
2h	1h	01h	MEDIA_OBJECT_EX	Media
2h	1h	02h	MEDIA_OBJECT_PRT	Media
2h	2h–7h	XX	Reserved	n/a
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	3D Pipeline
3h	0h	01h	3DSTATE_BINDING_TABLE_POINTERS	3D Pipeline
3h	0h	02h	Reserved	
3h	0h	03h–04h	Reserved	n/a
3h	0h	05h	3DSTATE_URB	3D Pipeline
3h	0h	06h-07h	Reserved	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	Reserved	n/a
3h	0h	0Ch	Reserved	n/a
3h	0h	0Dh	3DSTATE_VIEWPORT_STATE_POINTERS	3D Pipeline
3h	0h	0Eh–FFh	Reserved	n/a
3D State (Non-Pipelined)	**Bits 26:24**	**Bits 23:16**		
3h	1h	00h	3DSTATE_DRAWING_RECTANGLE	Strips & Fans
3h	1h	01h	3DSTATE_CONSTANT_COLOR	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	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	Windower
3h	1h	0Ah	3DSTATE_AA_LINE_PARAMS [DevCTG], [DevEL]	Windower
3h	1h	0Bh	3DSTATE_GS_SVB_INDEX [DevCTG], [DevEL]	Geometry Shader
3h	1h	0Ch	3DSTATE_SAMPLER_PALETTE_LOAD1	Sampling Engine
3h	1h	0Ah–FFh	Reserved	Windower
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 Register Address Maps

5.1 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.

5.1.1 Memory and I/O Space Registers

This section provides a high-level register map (register groupings per function). The memory and I/O maps for the graphics device registers are shown in the following table, except PCI Configuration registers that are described in the following chapter.

The VGA and Extended VGA registers can be accessed via standard VGA I/O locations as well as via memory-mapped locations.

All graphics MMIO registers can also be accessed via CPU I/O.

The memory space address listed for each register is an offset from the base memory address programmed into the MMADR register (PCI configuration offset 14h).

Table 5-1. Graphics Controller Register Memory and I/O Map

Start Offset	End Offset	Description
00000h	00FFFh	**VGA and Extended VGA Control Registers.** These registers are located in both I/O space and memory space. The VGA and Extended VGA registers contain the following register sets: General Control/Status, Sequencer (SRxx), Graphics Controller (GRxx), Attribute Controller (Arxx), VGA Color Palette, and CRT Controller (CRxx) registers. Detailed bit descriptions are provided in the *VGA and Extended VGA Register* Chapter. The registers within a set are accessed using an indirect addressing mechanism as described at the beginning of each section. Note that some of the register description sections have additional operational information at the beginning of the section
01000h	01FFFh	***Reserved***

Start Offset	End Offset	Description
02000h	02FFFh	**Instruction, Memory, and Interrupt Control Registers:** **Instruction Control Registers** Ring Buffer registers and page table control registers are located in this address range. Various instruction status, error, and operating registers are located in this group of registers. **Graphics Memory Fence Registers.** The Graphics Memory Fence registers are used for memory tiling capabilities. **Interrupt Control/Status Registers.** This register set provides interrupt control/status for various GC functions. **Display Interface Control Register.** This register controls the FIFO watermark and provides burst length control. **Logical Context Registers** **Pipeline Statistic Counters**
03000h	031FFh	**FENCE & Per Process GTT Control registers**
03200h	03FFFh	**Frame Buffer Compression Registers**
04000h	043FFh	***Reserved.***
04400h	04FFFh	***Reserved.***
05000h	05FFFh	**I/O Control Registers**
06000h	06FFFh	**Clock Control Registers**. This memory address space is the location of the GC clock control and power management registers
07000h	073FFh	**3D Internal Debug Registers**
07400h	088FFh	**GPE Debug Registers (3D/Media Fixed Functions)**
08900h	08FFFh	***Reserved*** **for Subsystem Debug Registers**
09000h	09FFFh	***Reserved***
0A000h	0AFFFh	**Display Palette Registers**
0B000h	0FFFFh	***Reserved***
10000h	13FFFh	**MMIO MCHBAR.** Alias through which the graphics driver can access registers in the MCHBAR accessed through device 0.
14000h	2FFFFh	***Reserved***
30000h	3FFFFh	**Overlay Registers.** These registers provide control of the overlay engine. The overlay registers are double-buffered with one register buffer located in graphics memory and the other on the device. On-chip registers are not directly writeable. To update the on-chip registers software writes to the register buffer area in graphics memory and instructs the device to update the on-chip registers.
40000h	5FFFFh	***Reserved***
60000h	6FFFFh	**Display Engine Pipeline Registers**
70000h	72FFFh	**Display and Cursor Registers**
73000h	73FFFh	**Performance Counters**
74000h	7FFFFh	***Reserved***

5.1.2 PCI Configuration Space

See the releveant EDS for details on accessing PCI configuration space, PCI address map tables, and register descriptions.

5.1.3 Graphics Register Memory Address Map

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 5-2 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	PGTBL_ER	Page Table Error Register (*DEBUG*)	RO
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	HW_MEMRD	Memory Read Sync Register (*DEBUG*)	RO
02064h–02067h	IPEIR	Instruction Parser Error Identification Register (*DEBUG*)	RO
02068h–0206Bh	IPEHR	Instruction Parser Error Header Register (*DEBUG*)	RO
0206Ch–0206Fh	INSTDONE	Instruction Stream Interface Done Register (*DEBUG*)	RO
02070h–02073h	INSTPS	Instruction Parser State Register (*DEBUG*)	RO

Table 5-2 Memory Mapped Registers

02074h–02077h	ACTHD	Active Head Pointer Register (*DEBUG*)	RO
02078h–0207Bh	DMA_FADD_P	Primary DMA Engine Fetch Address Register (*DEBUG*)	RO
0207Ch–0207Fh	INSTDONE_1	Instruction Stream Interface Done 1 (Debug)	RO
02080h–02083h	HWS_PGA	Hardware Status Page Address Register	R/W
02084h–02087h	—	Reserved	—
02088h–0208Ch	PWRCTXA	Power Context Register Address ([DevCL])	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	PGTBL_CTL2	Per-process Page Table Control 0 [DevBW], [DevCL] only	R/W
020C8h–020CBh	PGTBL_STR2	Page Table Steer Register (Per Process) [DevBW], [DevCL] only	R/W
020CCh–020DFh	—	Reserved	—
020E0h–020E3h	MI_DISPLAY_POWER_DOWN	Display Power Down Enable ([DevCL] Only)	R/W
	MI_RDRET_STATE	Memory Interface Read Return State Register ([DevBW] Only)	R/W
020E4h–020E7h	MI_ARB_STATE	Memory Interface Arbitration State Register (SAVED/RESTORED)	R/W
020E8h–020FBh	—	Reserved	—
020FCh–020FFh	MI_RDRET_STATE	Memory Interface Read Return State Register ([DevCL] Only)	R/W
02100h–0210Fh	—	Reserved	—

Table 5-2 Memory Mapped Registers

02110h–02113h	BB_STATE	Batch Buffer State Register	R/W
02114h–0211Fh	—	Reserved	—
02120h–02123h	CACHE_MODE_0	Cache Mode Register 0 (*DEBUG*) (SAVED/RESTORED)	R/W
02124h–02127h	CACHE_MODE_1	Cache Mode Register 1 (*DEBUG*) (SAVED/RESTORED)	R/W
02128h–02133h	—	Reserved	—
02134h–02137h	UHPTR	Pending Head Pointer Register	R/W
02138h–0213Fh	—	Reserved	—
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 [DevCTG], [DevEL]	R/W
0217Ch–0217Fh	PR_CTR_THRSH	Render Watchdog Counter Threshold [DevCTG], [DevEL]	R/W
02180h–02183h	CCID0	Current Context ID 0 (assoc w/ PRB0)	R/W
02184h–0218Fh	—	Reserved	—
02190h–02193h	PR_CTR	Render Watchdog Counter [DevCTG], [DevEL]	RO
02194h–0219Fh	—	Reserved	—
021A0h–021A3h	CXT_SIZE	Context Size (*DEBUG*)	R/W
021A4h–021A7h	CXT_SIZE_NOEXT	Context Size without Ext. State (*DEBUG*)	R/W
021A8h-021CFh	—	Reserved	—
021D0h-021D3h	ECOSKPD	ECO Scratch Pad (*DEBUG*)	R/W
021D4h-021FFh	—	Reserved	—
02200h–02303h	CSFLFSM	Flush FSM (Debug)	R/W
02204h–02207h	CSFLFLAG	Flush FLAG (Debug)	R/W
02208h–0220Bh	CSFLTRK	Flush Track (Debug)	R/W
0220Ch–0220Fh	CSCMDOP	Instruction DWORD (Debug)	R/W
02210h–02213h	CSCMDVLD	Instruction DWORD Valid (Debug)	R/W
02214h–0230Fh	—	Reserved	—
02310h-02347h	—	Reported Vertices Counter	R/W
02350h-02357h	PS_DEPTH_COUNT	Reported Pixels Passing Depth Test Counter	R/W

Table 5-2 Memory Mapped Registers

02358–0235Fh	TIMESTAMP	Reported Timestamp Count	R/W
02360–02367h	CLKCMP	Compare Count Clock Stop (Debug)	
02368h–0236Fh	—	Reserved	—
02370h–02377h	—	Reserved	—
02378h–0237Fh	—	Reserved	—
02380h–02387h	—	Reserved	—
02388h–0244Fh	—	Reserved	—
02450h–02453h	VFDC	Set Value of Draw Count (*DEBUG*)	R/W
02454h–0246Fh	—	Reserved	—
02470h–02473h	VFSKPD	VF Scratch Pad (*DEBUG*)	R/W
02474h–024FFh	—	Reserved	—
Per-Process GTT Control (02500h–025FFh)			
02500h–02503h	PP_DCIR	PPGTT Directory Cache Index Register (*DEBUG*)	R/W
02504h–02507h	PP_DCDR	PPGTT Directory Cache Data Register (*DEBUG*)	WO
02508h–0250Fh	PP_DCLV	PPGTT Directory Cacheline Valid Register (*DEBUG*)	R/W
02510h–02513h	PP_PFIR	PPGTT Page Fault Indication Register	R/WC
02514h–02517h	PP_PFIC	PPGTT Page Fault Interface Control Register	R/WC
02518h–0251Bh	PP_DIR_BASE	PPGTT Page Directory Base Register (*DEBUG*)	R/W
0251Ch–0251Fh	TLB_RD_EXT	TLB Read Extent	RO
02520h–02523h	GFX_MODE	Graphics Mode Register	R/W
02524h–0257Fh	—	Reserved	—
02580h–025FFh	PP_PFD[31:0]	PPGTT Page Fault Data (32, 1 DW each)	RO
Probe List Control (02600h–026FFh) : Reserved			
02600h–0267Fh	PRBL_SV	Probe List Slot Valid Registers (32, 1 DW each)	RO
02680h–02683h	PRBL_SFL	Probe List Slot Fault Low	RO
02684h–02687h	PRBL_SFH	Probe List Slot Fault High	RO
02688h–026FFh	—	Reserved	—
Run List Control (02700h–027FFh) : Reserved			
02700h–02703h	RLSP	Run List Submit Port	WO
02704h–02707h	RLS	Run List Status Register	RO

Table 5-2 Memory Mapped Registers

02708h–0270Bh	—	Reserved	—
0270Ch–0270Fh	CTXT_ST_PTR	Context Status Buffer Pointer Register (Debug)	R/W
02710h–02713h	—	Reserved	—
02714h–02717h	CTXT_SR_CTL	Context Save/Restore Control	R/W
02718h–0271Bh	CTXT_PREMP_DBG	Pre-emption Debug Register (Debug)	R/W
0271Ch–0271Fh	CTXT_WAIT_STS	Wait Status Register	RO
02720h–02723h	CTXT_COUNT	Context Invocation Count	R/W
02724h–02727h	GLBL_PREMP_CNT	Global Pre-emption Count	R/W
02728h–0272Fh	CTXT_EXEC_CUM	Context Cumulative Execution Time	R/W
02730h–02737h	CTXT_OVHD_CUM	Context Cumulative Overhead Time	R/W
02738h–0273Fh	GLBL_SWITCH_CUM	Global Context Switch Cumulative Time	R/W
02740h–027BFh	—	Reserved	—
027C0h–027DFh	RLC0[7:0]	Run List 0 Contents Registers (Debug, 8 DWs)	RO
027E0h–027FFh	RLC1[7:0]	Run List 1 Contents Registers (Debug, 8 DWs)	RO
02800h–02FFFh	—	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
Frame Buffer Compression Control (03200h–03FFFh) ([DevCL] Only)			
03200h–03203h	FBC_CFB_BASE	Compressed Frame Buffer Base Address	R/W
03204h–03207h	FBC_LL_BASE	Compressed Frame Line Length Buffer Address	R/W
03208h–0320Bh	FBC_CONTROL	Frame Buffer Compression Control Register	R/W
0320Ch–0320Fh	FBC_COMMAND	Frame Buffer Compression Command Register	R/W
03210h–03213h	FBC_STATUS	Frame Buffer Compression Status Register	R/W
03214h–03217h	FBC_CONTROL2	Frame Buffer Compression 2nd Control Register	R/W
0321Bh–0321Eh	FBC_DISPYOFF	Frame Buffer Compression Display Y Offset	R/W
03220h–03223h	FBC_MOD_NUM	Frame Buffer Compression Num of Modifications	R/W
03214h–032FFh	—	Reserved	—

Table 5-2 Memory Mapped Registers

03300h–033C3h	FBC_TAG	Frame Buffer Compression Tag Interface (Debug)	R/W
03400h–03FFFh	—	Reserved	—
Frame Buffer Compression Control (03200h–03FFFh) : [DevCTG]			
03200h–03203h	DPFC_CB_BASE	DPFC Compressed Buffer Base Address	R/W
03204h–03207h	—	Reserved	—
03208h–0320Bh	DPFC_CONTROL	DPFC Control	R/W
0320Ch–0320Fh	DPFC_RECOMP_CTL	DPFC ReComp Control	R/W
03210h–03213h	DPFC_STATUS	DPFC Status	RO
03214h–03217h	—	Reserved	—
03218h–0321Bh	DPFC_CPU_FENCE_OFFSET	DPFC CPU Fence Offset	R/W
0321Ch–0321Fh	DPFC_SLB_DATA	DPFC SLB Data	R/W
03220h–03223h	DPFC_DEBUG_STATUS	DPFC Debug Status	R/W
03224h–03227h	DPFC_CHICKEN	DPFC Chicken Bits	R/W
03228h–03FFFh	—	Reserved	—
03200h–03203h	DPFC_CB_BASE	DPFC Compressed Buffer Base Address	R/W
BCS Instruction and Interrupt Control Registers (04000h–043FFh)			
04000h–043FFh	—	Reserved	—
04064h–04067h	BCS_IPEIR	Instruction Parser Error Identification Register (Debug)	RO
04068h–0406Bh	BCS_IPEHR	Instruction Parser Error Header Register (Debug)	RO
04074h–04077h	BCS_ACTHD	Active Head Pointer Register (Debug)	RO
04078h – 0407Bh	BCS_DMA_FADD	DMA Engine Fetch Address (Debug)	RO
04080h–04083h	BCS_HWS_PGA	Hardware Status Page Address Register	R/W
04084h–04093h	—	Reserved	—
04094h–04097h	BCS_NOPID	NOP Identification Register	RO
04097h–0409B	—	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	R/W
04138h–04177h	—	Reserved	—

Table 5-2 Memory Mapped Registers

04178h–0417Bh	BCS_CNTR	Counter for the Bit Stream Decode Engine	R/W
0417Ch–0417Fh	BCS_THRSH	Threshold for the Counter of Bit Stream Decode Engine	R/W
04180h–0413Fh	—	Reserved	—
04140h–04147h	BCS_BB_ADDR	Batch Buffer Head Pointer Register	RO
04148h–0418Fh	—	Reserved	—
04190h–04193h	BCS_RCCID	Ring Buffer Current Context ID	R/W
04194h–04197h	BCS_RNCID	Ring Buffer Next Context ID	R/W
04198h–043FFh	—	Reserved	—
I/O Control Registers (05000h–05FFFh)			
05000h–0500Fh	—	Reserved	—
05010h–05013h	GPIO_CTL0	General Purpose I/O Control Register [0]	R/W
05014h–05017h	GPIO_CTL1	General Purpose I/O Control Register [1]	R/W
05018h–0501Bh	GPIO_CTL2	General Purpose I/O Control Register [2]	R/W
0501Ch–0501Fh	GPIO_CTL3	General Purpose I/O Control Register [3]	R/W
05020h–05023h	GPIO_CTL4	General Purpose I/O Control Register [4]	R/W
05024h–05027h	GPIO_CTL5	General Purpose I/O Control Register [5]	R/W
05028h–0502Bh	GPIO_CTL6	General Purpose I/O Control Register [6]	R/W
0502Ch–0502Fh	GPIO_CTL7	General Purpose I/O Control Register [7]	R/W
05030h–050FFh	—	Reserved	—
05100h–05103h	GMBUS0	GMBUS Clock Select/Device Select	R/W
05104h–05107h	GMBUS1	GMBUS Command/Status	R/W
05108h–0510Bh	GMBUS2	GMBUS Status	R/W
0510Ch–0510Fh	GMBUS3	GMBUS Data Buffer	R/W
05110h–05F13h	GMBUS4	GMBUS Interrupt Mask	R/W
05114h–0511Fh	—	Reserved	—
05120h–05123h	GMBUS5	GMBUS 2-Byte Index Register	R/W
05124h–05FFFh	—	Reserved	—
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

Table 5-2 Memory Mapped Registers

06018h–0601Bh	DPLLB_CTRL	Display PLL B Control	R/W
0601Ch–0601Fh	DPLLAMD	Display PLL A UDI Multiplier/Divsor	R/W
06020h–06023h	DPLLBMD	Display PLL B UDI Multiplier/Divsor	R/W
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	DPLL_TEST	DPLLA and DPLLB Test Register	R/W
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	R/W
0620Ch–0620Fh	VDECCLK_GATE_D	Clock Gating Disable for Video Decode Register ([DevCTG], [DevEL] Only)	R/W
06210h–06213h	RAMCLK_GATE_D	GFX RAM Clock Gating Disable Register ([DevCL] Only)	R/W
06214h–06125h	DEUC	Dynamic EU Control	R/W/L
06216h–06FFFh	—	Reserved	—
3D-Internal Debug Registers (07000h–073FFh) Reserved			
07000h–073FFh	—	Reserved	—
GPE Debug Registers (07400h–088FFh, DEBUG ONLY, Subject to Change)			
07400h–07403h	SVG_CTL	Debug Control	R/W
07404h–07407h	SVG_RDATA	Debug Return Data	RO
07408h–0740Bh	SVG_WORK_CTL	Debug Workaround Control	R/W
0740Ch–074FFh	—	Reserved	—
07500h–07503h	VF_CTL	Debug Control	R/W
07504h–07507h	VF_STRG_VAL	Debug Snapshot Trigger Value	R/W
07508h–0750Bh	VF_STR_VL_OVR	Debug Start Vertex Location Override	R/W
0750Ch–0750Fh	VF_VC_OVR	Debug Vertex Count Override	R/W
07510h–07513h	VF_STR_PSKIP	Debug Starting Primitives Skipped	RO

Table 5-2 Memory Mapped Registers

07514h–07517h	VF_MAX_PRIM	Debug Max Primitives	R/W
07518h–0751Bh	VF_RDATA	Debug Return Data	RO
0751Ch–075FFh	—	Reserved	—
07600h–07603h	VS_CTL	Debug Control	R/W
07604h–07607h	VS_STRG_VAL	Debug Snapshot Trigger Value	R/W
07608h–0760Bh	VS_RDATA	Debug Return Data	RO
0760Ch–078FFh	—	Reserved	—
07900h–07903h	GS_CTL	Debug Control	R/W
07904h–07907h	GS_STRG_VAL	Debug Snapshot Trigger Value	R/W
07908h–0790Bh	GS_RDATA	Debug Return Data	RO
0790Ch–079FFh	—	Reserved	—
07A00h–07A03h	CL_CTL	Debug Control	R/W
07A04h–07A07h	CL_STRG_VAL	Debug Snapshot Trigger Value	R/W
07A08h–07A0Bh	CL_RDATA	Debug Return Data	RO
07A0Ch–07AFFh	—	Reserved	—
07B00h–07B03h	SF_CTL	Debug Control	R/W
07B04h–07B07h	SF_STRG_VAL	Debug Snapshot Trigger Value	R/W
07B08h–07B0Bh	SF_MIN_PR_IND	Debug Minimum Primitive Index	R/W
07B0Ch–07B0Fh	SF_MAX_PR_IND	Debug Maximum Primitive Index	R/W
07B10h–07B13h	SF_CLIP_RMIN	Debug Clip Rectangle Minimum Coordinates	R/W
07B14h–07B17h	SF_CLIP_RMAX	Debug Clip Rectangle Maximum Coordinates	R/W
07B18h–07B1Bh	SF_RDATA	Debug Return Data	RO
07B1Ch–07BFFh	—	Reserved	—
07C00h–07C03h	WIZ_CTL	Debug Control	R/W
07C04h–07C07h	WIZ_STRG_VAL	Debug Snapshot Trigger Value	R/W
07C08h–07C0Bh	WIZ_RDATA	Debug Return Data	RO
07C0Ch–07CFFh	—	Reserved	—
07D00h–07D03h	VFE_CTL	Debug Control	R/W
07D04h–07D07h	VFE_STRG_VAL	Debug Snapshot Trigger Value	R/W
07D08h–07D0Bh	VFE_RDATA	Debug Return Data	RO
07D0Ch–07DFFh	—	Reserved	—
07E00h–07E03h	TS_CTL	Debug Control	R/W
07E04h–07E07h	TS_STRG_0-6VAL	Debug Snapshot Trigger R0.6 Value	R/W
07E08h–07E0Bh	TS_STRG_0-7VAL	Debug Snapshot Trigger R0.7 Value	R/W

Table 5-2 Memory Mapped Registers

07E0Ch–07E0Fh	TS_RDATA	Debug Return Data	RO
07E10h–07FFFh	—	Reserved	—
08000h–08003h	TD_CTL	Debug Control	R/W
08004h–08007h	TD_CTL2	Debug Control 2	R/W
08008h–0800Bh	TD_VF_VS_EMSK	Debug VF/VS Execution Mask	R/W
0800Ch–0800Fh	TD_GS_EMSK	Debug GS Execution Mask	R/W
08010h–08013h	TD_CLIP_EMSK	Debug Clipper Execution Mask	R/W
08014h–08017h	TD_SF_EMSK	Debug SF Execution Mask	R/W
08018h–0801Bh	TD_WIZ_EMSK	Debug WIZ Execution Mask	R/W
0801Ch–0801Fh	TD_0-6_EHTRG_VAL	Debug R0.6 External Halt Trigger Value	R/W
08020h–08023h	TD_0-7_EHTRG_VAL	Debug R0.7 External Halt Trigger Value	R/W
08024h–08027h	TD_0-6_EHTRG_MSK	Debug R0.6 External Halt Trigger Mask	R/W
08028h–0802Bh	TD_0-7_EHTRG_MSK	Debug R0.7 External Halt Trigger Mask	R/W
0802Ch–0802Fh	TD_RDATA	Debug Return Data	RO
08030h–08033h	TD_TS_EMSK	Debug TS Execution Mask	—
08034h–080FFh	—	Reserved	—
08100h–08103h	MATH_CTL	Math Debug Control	R/W
08104h–08107h	MATH_RDATA	Math Debug Return Data	RO
08108h–081FFh	—	Reserved	—
08200h–08203h	ISC_CTL	Instruction / State Debug Control	R/W
08204h–0827FFh	—	Reserved	—
08280h–08283h	ISC_L1CA_CTR	Instruction L1 Cache Debug Control	RO
08284h–08287h	ISC_L1CA_RDATA	Instruction L1 Cache Debug Return Data	
08288h–0828Bh	ISC_L1CA_BP_ADR1	Instruction L1 Cache Breakpoint Address 1 Control	
0828Ch–0828Fh	—	Reserved	—
08290h–08293h	ISC_L1CA_BP_ADR2	Instruction L1 Cache Breakpoint Address 2 Control	
08294h–08297h	ISC_L1CA_BP_OPC1	Instruction L1 Cache Breakpoint Opcode 1 Control	
08298h–0829Bh	ISC_L1CA_BP_OPC2	Instruction L1 Cache Breakpoint Opcode 2 Control	
0829Ch–082FFh	—	Reserved	—
08300h–08303h	MA_DEBUG_1	Message Arbiter Debug Control	R/W
08304h–083FFh	—	Reserved	—

Table 5-2 Memory Mapped Registers

08400h–08403h	SAMPLER_CTL	Sampler Debug Control	R/W
08404h–08407h	SAMPLER_RDATA	Sampler Debug Return Data	RO
08408h–084FFh	—	Reserved	—
08500h–08503h	DP_CTL	Data Port Debug Control	R/W
08504h–08507h	DP_RDATA	Data Port Debug Return Data	RO
08508h–085FFh	—	Reserved	—
08600h–08603h	RC_CTL	Debug Control	R/W
08604h–08607h	RC_DEF_CLR	Debug Force Default Color	R/W
08608h–0860Bh	RC_RDATA	Debug Return Data	RO
0860Ch–086FFh	—	Reserved	—
08700h–08703h	URB_CTL	Debug Control	R/W
08704h–08707h	—	Reserved	—
08708h–0870Bh	URB_RDATA	Debug Return Data	RO
0870Ch–087FFh	—	Reserved	—
08800h–08803h	EU_CTL	Debug Control	R/W
08804h–0880Fh	—	Reserved	—
08810h–08817h	EU_ATT	Debug Attention	RO
08818h–0881Fh	—	Reserved	—
08820h–08827h	EU_ATT_DATA	EU Debug Attention Data	RO
08828h–0882Fh	—	Reserved	—
08830h–08837h	EU_ATT_CLR	Debug Attention Clear	WO
08838h–0883Fh	—	Reserved	—
08840h–08843h	EU_RDATA	Debug Return Data	RO
08844h–088FFh	—	Reserved	—
Reserved for Debug (08900h–09FFFh)			
08900h–08FFFh	—	Reserved for Subsystem Debug	—
09000h–09FFFh	—	Reserved	—
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	—

Table 5-2 Memory Mapped Registers

TLB Read Range (0B000h–0BFFFh) : Reserved			
0B000h–0BFFFh	—	Reserved	—
AVC Video Decode (0C000h–0CFFFh) : Reserved			
0C000h-0CFFFh	--	Reserved	--
0D000h–0FFFFh	—	Reserved	—
GFX MMIO – MCHBAR Aperture (10000h-13FFFh)			
10000h-13FFFh	—	MCHBAR Aperture	R/W
Reserved (14000h–2FFFFh)			
14000h-2FFFFh	—	Reserved	—
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	OTEST	Overlay Test Register	R/W
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

Table 5-2 Memory Mapped Registers

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
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

Table 5-2 Memory Mapped Registers

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	—
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	BCLRPAT_A	Pipe A Border Color Pattern	R/W
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

Table 5-2 Memory Mapped Registers

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
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	BCLRPAT_B	Pipe B Border Color Pattern	R/W
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	Reserved	Reserved	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	Reserved	R/W
6112Ch–6112Fh	—	Reserved	—

Table 5-2 Memory Mapped Registers

61130h–61133h	Reserved	Reserved	R/W
61134h–61137h	Reserved	Reserved	R/W
61138h–6113Bh	UDID	Reserved	R/W
6113Ch–6113Fh	—	Reserved	—
61140h-61143h	Reserved	Reserved	R/W
61144h–61147h	Reserved	Reserved	R/W
61148h–6114Bh	Reserved	Reserved	R/W
6114Ch–6114Fh	Reserved	Reserved	R/W
61150h-61153h	Reserved	Reserved	R/W
61154h–61157h	Reserved	Reserved	R/W
61158h–6115Bh	Reserved	Reserved	R/W
6115Ch–6115Fh	Reserved	Reserved	
61160h-61163h	Reserved	Reserved	R/W
61164h–61167h	Reserved	Reserved	R/W
61168h–6116Bh	Reserved	Reserved	R/W
6116Ch–6116Fh	—	Reserved	—
61170h–61173h	UDI_IF_CTL	UDI InfoFrame Control	R/W
61174h–61177h	—	Reserved	—
61178h–6117Bh	UDI_VIDPAC_DATA	UDI Video-related Data Island Packet Data	R/W
6117Ch–61177h	—	Reserved	—
LVDS ([DevCL] Only)			
61180h–61183h	LVDS	Reserved	R/W
61184h–611FFh	—	Reserved	—
Panel Power Sequencing ([DevCL] Only)			
61200h–61203h	PP_STATUS	Panel Power Status	RO
61204h–61207h	PP_CONTROL	Panel Power Control	R/W
61208h–6120Bh	PP_ON_DELAYS	Panel Power On Sequencing Delays	R/W
6120Ch–6120Fh	PP_OFF_DELAYS	Panel Power Off Sequencing Delays	R/W
61210h–61213h	PP_DIVISOR	Panel Power Cycle Delay and Reference Divisor	R/W

Table 5-2 Memory Mapped Registers

61214h–6122Fh	—	Reserved	—
Panel Fitting ([DevCL] Only)			
61230h–61233h	PFIT_CONTROL	Panel Fitting Control	R/W
61234h–61237h	PFIT_PGM_RATIOS	Programmed Panel Fitting Ratios	R/W
61238h–6124Fh	—	Reserved	—
Backlight Control and Modulation ([DevCL] Only)			
61250h–61253h	BLC_PWM_CTL2	Backlight PWM Control Register 2	R/W
61254h–61257h	BLC_PWM_CTL	Backlight PWM Control	R/W
61258h–6125Fh	—	Reserved	—
61260h–61263h	BLM_HIST_CTL	Image BLM Histogram Control Register	R/W
61264h–61267h		Image Enhancement Bin Data Register	RO, R/W
61268h–6126Bh		Histogram Threshold Guardband Register	R/W
6126Ch–61FFFh	—	Reserved	—
High Definition Registers (62000h–62FFFh)			
62000h–62003h	Reserved	Reserved	R/W
62004h–6200Fh	Reserved	Reserved	—
62010h–62013h	Reserved	Reserved	RO
62014h–6201Fh	Reserved	Reserved	—
62020h–62023h	Reserved	Reserved	RO
62024h–62027h	Reserved	Reserved	RO
62028h–6202Bh	Reserved	Reserved	RO
6202Ch–6203Fh	Reserved	Reserved	—
62040h–62043h	Reserved	Reserved	RO
62044h–62047h	Reserved	Reserved	RO
62048h–6204Bh	Reserved	Reserved	RO
6204Ch–6204Fh	Reserved	Reserved	RO
62050h–62053h	Reserved	Reserved	RO
62054h–62057h	Reserved	Reserved	RO
62058h–6206Fh	Reserved	Reserved	—

Table 5-2 Memory Mapped Registers

62070h–62073h	Reserved	Reserved	RO
62074h–62077h	Reserved	Reserved	R/W
62078h–6207Bh	Reserved	Reserved	R/W
6207Ch–6207Fh	Reserved	Reserved	R/W
62080h–62083h	Reserved	Reserved	RO
62084h–62087h	Reserved	Reserved	RO
62088h–6209Fh	Reserved	Reserved	—
620A0h–620A3h	Reserved	Reserved	RO
620A4h–620A7h	Reserved	Reserved	RO
620A8h–620ABh	Reserved	Reserved	RO
620ACh–620AFh	Reserved	Reserved	RO
620B0h–620B3h	Reserved	Reserved	RO
620B4h–620B7h	Reserved	Reserved	RO
620B8h–620BBh	Reserved	Reserved	RO
620BCh–620BFh	Reserved	Reserved	RO
620C0h–620D3h	Reserved	Reserved	—
620D4h–620D7h	Reserved	Reserved	R/W
620D8h–6210Bh	Reserved	Reserved	—
6210Ch–62117h	Reserved	Reserved	R/W
62118h–62127h	Reserved	Reserved	R/W
62128h–67FFFh	—	Reserved	—
TV Out Control Registers (68000h–6FFFFh) [CLN] only			
68000h–68003h	TV_CTL	TV Out Control	R/W
68004h-68007h	TV_DAC	TV DAC Control/Status	R/W, RO
68008h–6800Fh	—	Reserved	—
68010h–68013h	TV_CSC_Y	Color Space Convert Y	R/W
68014h–68017h	TV_CSC_Y2	Color Space Convert Y2	R/W
68018h–6801Bh	TV_CSC_U	Color Space Convert U	R/W

Table 5-2 Memory Mapped Registers

6801Ch–6801Fh	TV_CSC_U2	Color Space Convert U2	R/W
68020h–68023h	TV_CSC_V	Color Space Convert V	R/W
68024h–68027h	TV_CSC_V2	Color Space Convert V2	R/W
68028h–6802Bh	TV_CLR_KNOBS	Color Knobs	R/W
6802Ch–6802Fh	TV_CLR_LEVEL	Color Level Control	R/W
68030h–68033h	TV_H_CTL_1	H Control 1	R/W
68034h–68037h	TV_H_CTL_2	H Control 2	R/W
68038h–6803Bh	TV_H_CTL_3	H Control 3	R/W
6803Ch–6803Fh	TV_V_CTL_1	V Control 1	R/W
68040h–68043h	TV_V_CTL_2	V Control 2	R/W
68044h–68047h	TV_V_CTL_3	V Control 3	R/W
68048h–6804Bh	TV_V_CTL_4	V Control 4	R/W
6804Ch–6804Fh	TV_V_CTL_5	V Control 5	R/W
68050h–68053h	TV_V_CTL_6	V Control 6	R/W
68054h–68057h	TV_V_CTL_7	V Control 7	R/W
68058h–6805Fh	—	Reserved	—
68060h–68063h	TV_SC_CTL_1	SubCarrier Control 1	R/W
68064h–68067h	TV_SC_CTL_2	SubCarrier Control 2	R/W
68068h–6806Bh	TV_SC_CTL_3	SubCarrier Control 3	R/W
6806Ch–6806Fh	—	Reserved	—
68070h–68073h	TV_WIN_POS	Window Position	R/W
68074h–68077h	TV_WIN_SIZE	Window Size	R/W
68078h–6807Fh	—	Reserved	—
68080h–68083h	TV_FILTER_CTL_1	Filter Control 1	R/W
68084h–68087h	TV_FILTER_CTL_2	Filter Control 2	R/W
68088h–6808Bh	TV_FILTER_CTL_3	Filter Control 3	R/W
6808Ch-6808Fh	SIN_ROM	Sine ROM	—
68090h-68093h	TV_CC_ CTL	Closed Caption Control	R/W
68094h-68097h	TV_CC_DATA1	Closed Caption Data Field 1	R/W
68098h-6809Bh	TV_CC_DATA2	Closed Caption Data Field 2	R/W
6809Ch–680AFh	—	Reserved	—
680B0h-680B3h	TV_WSS_ CTL	WSS Control	R/W
680B4h-680B7h	TV_WSS_DATA	WSS Data	R/W
68100h–681EFh	TV_H_LUMA	H Filter Luma Coefficients	R/W
681F0h–681FFh	—	Reserved	—
68200h–682EFh	TV_H_CHROMA	H Filter Chroma Coefficients	R/W
682F0h–682FFh	—	Reserved	—
68300h–683ABh	TV_V_LUMA	V Filter Luma Coefficients	R/W

Table 5-2 Memory Mapped Registers

683ACh–683FFh	—	Reserved	—
68400h–684ABh	TV_V_CHROMA	V Filter Chroma Coefficients	R/W
684ACh–6FFFFh	—	Reserved	—
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
70044h-70047h	PIPEAFRAMEPIX	Pipe A Frame Count Low and Pixel Count	RO
70048h-7007Fh	—	Reserved	—
Cursor A and 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

Table 5-2 Memory Mapped Registers

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	SWFxx	Software Flag 00:0F	R/W
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

Table 5-2 Memory Mapped Registers

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
71450h–71FFFh	—	Reserved	—
Display C / Sprite Control ([DevBW] and [DevCL])			
72000h–7217Fh	—	Reserved	—
72180h–72183h	DSPCCNTR	Display C / Sprite Control	R/W
72184h–72187h	DSPCLINOFF	Display C / Sprite Linear Offset Register	R/W
72188h–7218Bh	DSPCSTRIDE	Display C / Sprite Stride	R/W
7218Ch–7218Fh	DSPCPOS	Display C / Sprite Position	R/W
72190h–72193h	DSPCSIZE	Display C / Sprite Height and Width	R/W
72194h–72197h	DSPCKEYMINVAL	Display C / Sprite Color Key Min Value	R/W
72198h–7219Bh	DSPCKEYMSK	Display C / Sprite Color Key Mask	R/W
7219Ch–7219Fh	DSPCSURF	Display C Surface Address Register	R/W
721A0h–721A3h	DSPCKEYMAXVAL	Display C / Sprite Color Key Max Value	R/W
721A4h–721A7h	DSPCTILEOFF	Display C Tiled Offset Register	R/W
721A4h-721FFh	—	Reserved	—
72200h-72203h	DSPCFLPQSTAT	Flip Queue Status Register	R/W
72204h–721CFh	—	Reserved	—
721D0h–721D3h	DCLRC0	Display C Color Correction 0	R/W
721D4h–721D7h	DCLRC1	Display C Color Correction 1	R/W
721D8h–721DFh	—	Reserved	—

Table 5-2 Memory Mapped Registers

721E0h–721E3h	GAMC5	Display C Gamma Correction Register 5	R/W
721E4h–721E7h	GAMC4	Display C Gamma Correction Register 4	R/W
721E8h–721EBh	GAMC3	Display C Gamma Correction Register 3	R/W
721ECh–721EFh	GAMC2	Display C Gamma Correction Register 2	R/W
721F0h–721F3h	GAMC1	Display C Gamma Correction Register 1	R/W
721F4h–721F7h	GAMC0	Display C Gamma Correction Register 0	R/W
721F8h–723FFh	—	Reserved	—
Video Sprite A Control : (DevCTG)			
72000h–7217Fh	—	Reserved	—
72180h–72183h	DVSACNTR	Video Sprite A Control	R/W
72184h–72187h	DVSALINOFF	Video Sprite A Linear Offset	R/W
72188h–7218Bh	DVSASTRIDE	Video Sprite A Stride	R/W
7218Ch–7218Fh	DVSAPOS	Video Sprite A Position	R/W
72190h–72193h	DVSASIZE	Video Sprite A Height and Width	R/W
72194h–72197h	DVSAKEYVAL	Video Sprite A Color Key Value	R/W
72198h–7219Bh	DVSAKEYMSK	Video Sprite A Color Key Mask	R/W
7219Ch–7219Fh	DVSASURF	Video Sprite A Surface Address	R/W
721A0h–721A3h	DVSAKEYMAXVAL	Video Sprite A Color Key Max Value	R/W
721A4h–721A7h	DVSATILEOFF	Video Sprite A Tiled Offset	R/W
721A8h-721DFh	—	Reserved	—
721E0h–721E3h	DVSA_GAMC5	Video Sprite A Gamma Correction Register 5	R/W
721E4h–721E7h	DVSA_GAMC4	Video Sprite A Gamma Correction Register 4	R/W
721E8h–721EBh	DVSA_GAMC3	Video Sprite A Gamma Correction Register 3	R/W
721ECh–721EFh	DVSA_GAMC2	Video Sprite A Gamma Correction Register 2	R/W
721F0h–721F3h	DVSA_GAMC1	Video Sprite A Gamma Correction Register 1	R/W
721F4h–721F7h	DVSA_GAMC0	Video Sprite A Gamma Correction Register 0	R/W
721F8h–723FFh	—	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	—

Table 5-2 Memory Mapped Registers

Performance Counters (73000h-73FFFh)			
73000h–73003h	PCSRC	Performance Counter Source Register	R/W
73004h–73007h	PCSTAT	Performance Counter Status Register	RO
73008h–7317Fh	—	Reserved	—
Video Sprite B Control : Reserved			
73180h–73183h	DVSBCNTR	Video Sprite B Control	R/W
73184h–73187h	DVSBLINOFF	Video Sprite B Linear Offset	R/W
73188h–7318Bh	DVSBSTRIDE	Video Sprite B Stride	R/W
7318Ch–7318Fh	DVSBPOS	Video Sprite B Position	R/W
73190h–73193h	DVSBSIZE	Video Sprite B Height and Width	R/W
73194h–73197h	DVSBKEYVAL	Video Sprite B Color Key Value	R/W
73198h–7319Bh	DVSBKEYMSK	Video Sprite B Color Key Mask	R/W
7319Ch–7319Fh	DVSBSURF	Video Sprite B Surface Address	R/W
731A0h–731A3h	DVSBKEYMAXVAL	Video Sprite B Color Key Max Value	R/W
731A4h–731A7h	DVSBTILEOFF	Video Sprite B Tiled Offset	R/W
731A8h-731DFh	—	Reserved	—
731E0h–731E3h	DVSB_GAMC5	Video Sprite B Gamma Correction Register 5	R/W
731E4h–731E7h	DVSB_GAMC4	Video Sprite B Gamma Correction Register 4	R/W
731E8h–731EBh	DVSB_GAMC3	Video Sprite B Gamma Correction Register 3	R/W
731ECh–731EFh	DVSB_GAMC2	Video Sprite B Gamma Correction Register 2	R/W
731F0h–731F3h	DVSB_GAMC1	Video Sprite B Gamma Correction Register 1	R/W
731F4h–731F7h	DVSB_GAMC0	Video Sprite B Gamma Correction Register 0	R/W
731F8h–733FFh	—	Reserved	—
Reserved (74000h-7FFFFh)			
74000h–7FFFFh	—	Reserved	—

5.2 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.

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

Table 5-3. 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)
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

Address	Register Name (Read)	Register Name (Write)
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

5.3 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. Additional details concerning the indirect access mechanism are provided in the *VGA and Extended VGA Register Description* Chapter (see SRxx, GRxx, ARxx or CRxx sections).

Table 5-4. 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 5-5. 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 5-6. 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
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 5-7. 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
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	Test Register for Toggle State of Attribute Control Register

§§

6 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.

6.1 Memory Object Overview

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

6.1.1 Memory Object Types

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

Table 6-1. Memory Object Types

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.

Memory Object Type	Role
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.
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.

6.2 Channel Formats

6.2.1 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.

6.2.2 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.

6.2.3 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.

6.2.4 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.

6.2.5 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.

6.2.6 Floating Point (FLOAT)

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

6.2.6.1 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)

6.2.6.2 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)

6.3 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.

6.3.1 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.

6.3.2 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.

6.3.3 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.

6.3.4 P4A4_UNORM

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

7	4	3	0
Alpha		Palette Index	

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

6.3.5 A4P4_UNORM

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

7	4	3	0
Palette Index		Alpha	

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

6.3.6 P8A8_UNORM

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

15 8	7 0
Alpha	Palette Index

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

6.3.7 A8P8_UNORM

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

15 8	7 0
Palette Index	Alpha

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

6.3.8 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

6.3.9 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

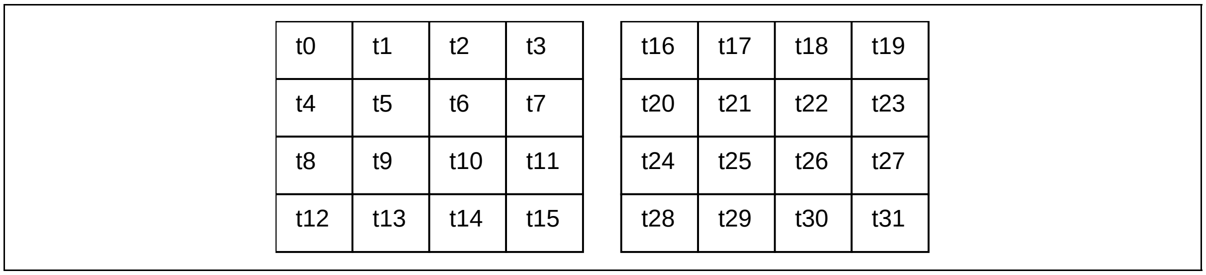
6.4 Compressed Surface Formats

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

6.4.1 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 6-1. FXT1 Encoded Blocks



t0	t1	t2	t3	t16	t17	t18	t19
t4	t5	t6	t7	t20	t21	t22	t23
t8	t9	t10	t11	t24	t25	t26	t27
t12	t13	t14	t15	t28	t29	t30	t31

6.4.1.1 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 6-2. FXT1 Format Summary

Bit 12 7	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.

6.4.1.2 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.

6.4.1.2.1 CC_HI Block Encoding

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

Table 6-3. 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

6.4.1.2.2 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 6-4. 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 6-5. 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
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.

6.4.1.3 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).

6.4.1.3.1 CC_CHROMA Block Encoding

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

Table 6-6. 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

Bit	Description
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

6.4.1.3.2 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 6-7. 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]

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
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 6-8. FXT CC_CHROMA Interpolated Color Table

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

6.4.1.4 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).

6.4.1.4.1 CC_MIXED Block Encoding

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

Table 6-9. 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

Bit	Description
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

6.4.1.4.2 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 6-10. 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 6-11. 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]

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
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]]
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 tables:

Table 6-12. 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 6-13. 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 6-14. 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 6-15. 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]

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
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]	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 6-16. 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 6-17. 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.

6.4.1.5 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.

6.4.1.5.1 CC_ALPHA Block Encoding

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

Table 6-18. 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

6.4.1.5.2 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 6-19. 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 6-20. 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 6-21. 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 6-22. 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

6.4.2 BC Texture Formats

The hardware supports five “BCn” surface formats that divide surfaces (texture maps) into independent 4x4 texel blocks and stores compressed versions of these blocks in 1 or 2 QWord units. Note that non-power-of-2 dimensioned maps may require the surface to be padded out to the next multiple of four texels – here the pad texels are not referenced by the device.

An 8-byte (QWord) block encoding can be used if the source texture contains no transparency (is opaque) or if the transparency can be specified by a one-bit alpha. A 16-byte (DQWord) block encoding can be used to support source textures that require more than one-bit alpha: here the 1st QWord is used to encode the texel alpha values, and the 2nd QWord is used to encode the texel color values.

These three types of format are discussed in the following sections:

Opaque and One-bit Alpha Textures (BC1)

Opaque Textures (BC1_RGB)

Textures with Alpha Channels (BC2-3)

Notes:

Any single texture must specify that its data is stored as 64 or 128 bits per group of 16 texels. If 64-bit blocks—that is, format BC1—are used for the texture, it is possible to mix the opaque and one-bit alpha formats on a per-block basis within the same texture. In other words, the comparison of the unsigned integer magnitude of color_0 and color_1 is performed uniquely for each block of 16 texels.

When 128-bit blocks are used, then the alpha channel must be specified in either explicit (format BC2) or interpolated mode (format BC3) for the entire texture. Note that as with color, once interpolated mode is selected then either 8 interpolated alphas or 6 interpolated alphas mode can be used on a block-by-block basis. Again the magnitude comparison of alpha_0 and alpha_1 is done uniquely on a block-by-block basis.

6.4.2.1 Opaque and One-bit Alpha Textures (BC1)

Texture format BC1 is for textures that are opaque or have a single transparent color. For each opaque or one-bit alpha block, two 16-bit R5G6B5 values and a 4x4 bitmap with 2-bits-per-pixel are stored. This totals 64 bits (1 QWord) for 16 texels, or 4-bits-per-texel.

In the block bitmap, there are two bits per texel to select between the four colors, two of which are stored in the encoded data. The other two colors are derived from these stored colors by linear interpolation.

The one-bit alpha format is distinguished from the opaque format by comparing the two 16-bit color values stored in the block. They are treated as unsigned integers. If the first color is greater than the second, it implies that only opaque texels are defined. This means four colors will be used to represent the texels. In four-color encoding, there are two derived colors and all four colors are equally distributed in RGB color space. This format is analogous to R5G6B5 format. Otherwise, for one-bit alpha transparency, three colors are used and the fourth is reserved to represent transparent texels. Note that the color blocks in BC2-3 strictly use four colors, as the alpha values are obtained from the alpha block.

In three-color encoding, there is one derived color and the fourth two-bit code is reserved to indicate a transparent texel (alpha information). This format is analogous to A1R5G5B5, where the final bit is used for encoding the alpha mask.

The following piece of pseudo-code illustrates the algorithm for deciding whether three- or four-color encoding is selected:

```
if (color_0 > color_1)
{
    // Four-color block: derive the other two colors.
    // 00 = color_0, 01 = color_1, 10 = color_2, 11 = color_3
    // These two bit codes correspond to the 2-bit fields
    // stored in the 64-bit block.
     color_2 = (2 * color_0 + color_1) / 3;
     color_3 = (color 0 + 2 * color_1) / 3;
}
else
{
    // Three-color block: derive the other color.
    // 00 = color_0,  01 = color_1,  10 = color_2,
    // 11 = transparent.
    // These two bit codes correspond to the 2-bit fields
    // stored in the 64-bit block.
     color_2 = (color_0 + color_1) / 2;
     color_3 = transparent;
}
```

The following tables show the memory layout for the 8-byte block. It is assumed that the first index corresponds to the y-coordinate and the second corresponds to the x-coordinate. For example, Texel[1][2] refers to the texture map pixel at (x,y) = (2,1).

Here is the memory layout for the 8-byte (64-bit) block:

Word Address	16-bit Word
0	Color_0
1	Color_1
2	Bitmap Word_0
3	Bitmap Word_1

Color_0 and Color_1 (colors at the two extremes) are laid out as follows:

Bits	Color
15:11	Red color component
10:5	Green color component
4:0	Blue color component

Bitmap Word_0 is laid out as follows:

Bits	Texel
1:0 (LSB)	Texel[0][0]
3:2	Texel[0][1]
5:4	Texel[0][2]
7:6	Texel[0][3]
9:8	Texel[1][0]
11:10	Texel[1][1]
13:12	Texel[1][2]
15:14	Texel[1][3]

Bitmap Word_1 is laid out as follows:

Bits	Texel
1:0 (LSB)	Texel[2][0]
3:2	Texel[2][1]
5:4	Texel[2][2]
7:6	Texel[2][3]
9:8	Texel[3][0]
11:10	Texel[3][1]
13:12	Texel[3][2]
15:14 (MSB)	Texel[3][3]

Example of Opaque Color Encoding

As an example of opaque encoding, we will assume that the colors red and black are at the extremes. We will call red color_0 and black color_1. There will be four interpolated colors that form the uniformly distributed gradient between them. To determine the values for the 4x4 bitmap, the following calculations are used:

```
00 ? color_0
01 ? color_1
10 ? 2/3 color_0 + 1/3 color_1
11 ? 1/3 color_0 + 2/3 color_1
```

Example of One-bit Alpha Encoding

This format is selected when the unsigned 16-bit integer, color_0, is less than the unsigned 16-bit integer, color_1. An example of where this format could be used is leaves on a tree to be shown against a blue sky. Some texels could be marked as transparent while three shades of green are still available for the leaves. Two of these colors fix the extremes, and the third color is an interpolated color.

The bitmap encoding for the colors and the transparency is determined using the following calculations:

```
00 ? color_0
01 ? color_1
10 ? 1/2 color_0 + 1/2 color_1
11   ?   Transparent
```

6.4.2.2 Opaque Textures (BC1_RGB)

Texture format BC1_RGB is identical to BC1, with the exception that the One-bit Alpha encoding is removed. Color 0 and Color 1 are not compared, and the resulting texel color is derived strictly from the Opaque Color Encoding. The alpha channel defaults to 1.0.

6.4.2.3 Compressed Textures with Alpha Channels (BC2-3)

There are two ways to encode texture maps that exhibit more complex transparency. In each case, a block that describes the transparency precedes the 64-bit block already described. The transparency is either represented as a 4x4 bitmap with four bits per pixel (explicit encoding), or with fewer bits and linear interpolation analogous to what is used for color encoding.

The transparency block and the color block are laid out as follows:

Word Address	64-bit Block
3:0	Transparency block
7:4	Previously described 64-bit block

Explicit Texture Encoding

For explicit texture encoding (BC2 formats), the alpha components of the texels that describe transparency are encoded in a 4x4 bitmap with 4 bits per texel. These 4 bits can be achieved through a variety of means such as dithering or by simply using the 4 most significant bits of the alpha data. However they are produced, they are used just as they are, without any form of interpolation.

Note:

DirectDraw’s compression method uses the 4 most significant bits.

The following tables illustrate how the alpha information is laid out in memory, for each 16-bit word.

This is the layout for Word 0:

Bits	Alpha
3:0 (LSB)	[0][0]
7:4	[0][1]
11:8	[0][2]
15:12 (MSB)	[0][3]

This is the layout for Word 1:

Bits	Alpha
3:0 (LSB)	[1][0]
7:4	[1][1]
11:8	[1][2]
15:12 (MSB)	[1][3]

This is the layout for Word 2:

Bits	Alpha
3:0 (LSB)	[2][0]
7:4	[2][1]
11:8	[2][2]
15:12 (MSB)	[2][3]

This is the layout for Word 3:

Bits	Alpha
3:0 (LSB)	[3][0]
7:4	[3][1]
11:8	[3][2]
15:12 (MSB)	[3][3]

Three-Bit Linear Alpha Interpolation

The encoding of transparency for the BC3 formats is based on a concept similar to the linear encoding used for color. Two 8-bit alpha values and a 4x4 bitmap with three bits per pixel are stored in the first eight bytes of the block. The representative alpha values are used to interpolate intermediate alpha values. Additional information is available in the way the two alpha values are stored. If alpha_0 is greater than alpha_1, then six intermediate alpha values are created by the interpolation. Otherwise, four intermediate alpha values are interpolated between the specified alpha extremes. The two additional implicit alpha values are 0 (fully transparent) and 255 (fully opaque).

The following pseudo-code illustrates this algorithm:

```
// 8-alpha or 6-alpha block?
if (alpha_0 > alpha_1) {
     // 8-alpha block:  derive the other 6 alphas.
    // 000 = alpha_0, 001 = alpha_1, others are interpolated
    alpha_2 = (6 * alpha_0 + alpha_1) / 7;      // bit code 010
    alpha_3 = (5 * alpha_0 + 2 * alpha_1) / 7;  // Bit code 011
    alpha_4 = (4 * alpha_0 + 3 * alpha_1) / 7;  // Bit code 100
    alpha_5 = (3 * alpha_0 + 4 * alpha_1) / 7;  // Bit code 101
    alpha_6 = (2 * alpha_0 + 5 * alpha_1) / 7;  // Bit code 110
    alpha_7 = (alpha_0 + 6 * alpha_1) / 7;      // Bit code 111
  }
else {  // 6-alpha block:  derive the other alphas.
    // 000 = alpha_0, 001 = alpha_1, others are interpolated
    alpha_2 = (4 * alpha_0 + alpha_1) / 5;      // Bit code 010
    alpha_3 = (3 * alpha_0 + 2 * alpha_1) / 5;  // Bit code 011
    alpha_4 = (2 * alpha_0 + 3 * alpha_1) / 5;  // Bit code 100
    alpha_5 = (alpha_0 + 4 * alpha_1) / 5;      // Bit code 101
    alpha_6 = 0;                                // Bit code 110
    alpha_7 = 255;                              // Bit code 111
}
```

The memory layout of the alpha block is as follows:

Byte	Alpha
0	Alpha_0
1	Alpha_1
2	[0][2] (2 LSBs), [0][1], [0][0]
3	[1][1] (1 LSB), [1][0], [0][3], [0][2] (1 MSB)
4	[1][3], [1][2], [1][1] (2 MSBs)
5	[2][2] (2 LSBs), [2][1], [2][0]
6	[3][1] (1 LSB), [3][0], [2][3], [2][2] (1 MSB)
7	[3][3], [3][2], [3][1] (2 MSBs)

6.4.3 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
}
```

6.4.4 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

Bit	Description
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.

6.5 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.

6.5.1 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 (UYVY) (R8G8_B8G8_UNORM)
- YCRCB_SWAPUVY (YUY2) (G8R8_G8B8_UNORM)
- YCRCB_SWAPUV
- YCRCB_SWAPY

The channels are mapped as follows:

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

Figure 6-2. Memory Layout of Packed YUV 4:2:2 Formats

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

YUV 4:2:2 (Normal)

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

YUV 4:2:2 (UV Swap)

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

YUV 4:2:2 (Y Swap)

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

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

Mem_Layout_YUV 422

6.5.2 Planar Memory Organization

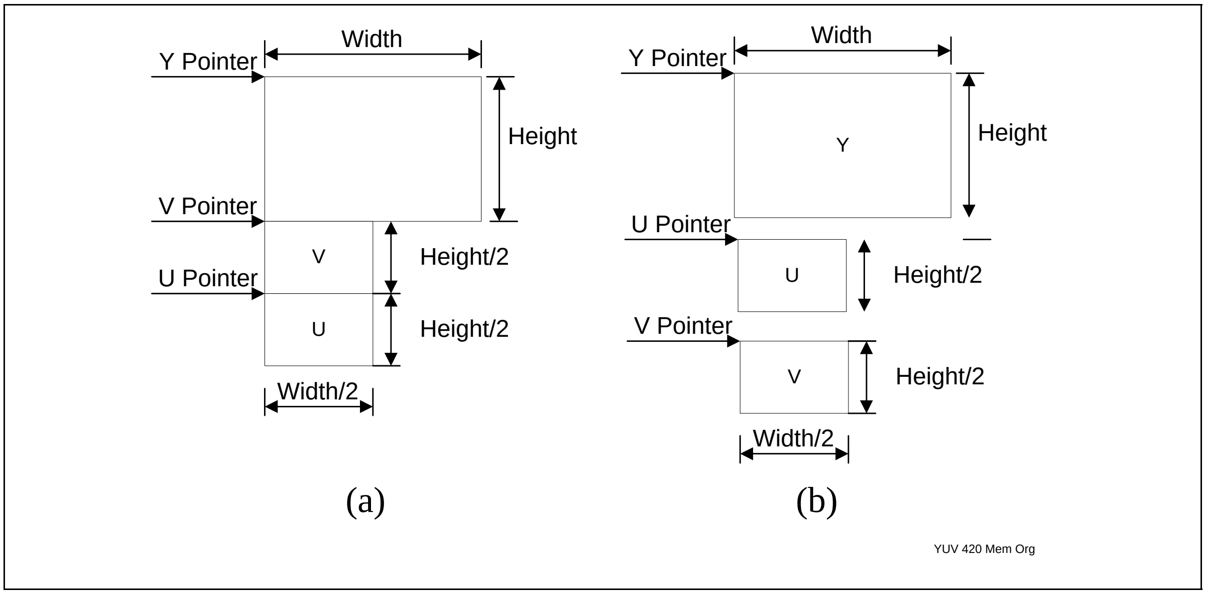
Planar formats use what could be thought of as separate buffers for the three color components. Because there is a separate stride for the Y and U/V data buffers, several memory footprints can be supported.

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:

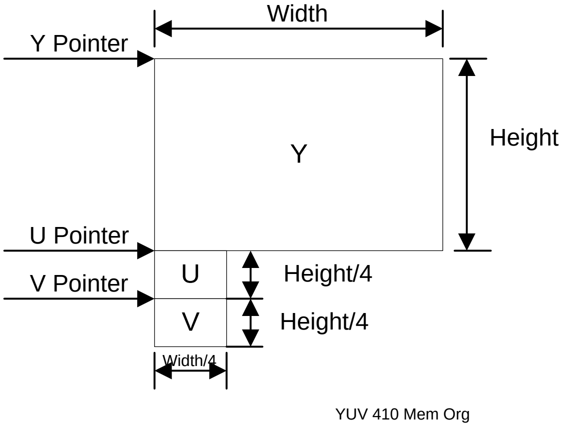
1. The memory organization of the common YV12 data, where all three planes are contiguous and the strides of U and V components are half of that of the Y component.
2. An alternative memory structure that the addresses of the three planes are independent but satisfy certain alignment restrictions.

Figure 6-3. 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 6-4. YUV 4:1:0 Format Memory Organization



6.6 Surface Memory Organizations

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

6.7 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.

6.8 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.

6.9 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.

6.10 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).

6.11 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.

6.12 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.

6.13 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.

6.14 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.

6.15 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.

6.16 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 6-23. 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 upper 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.

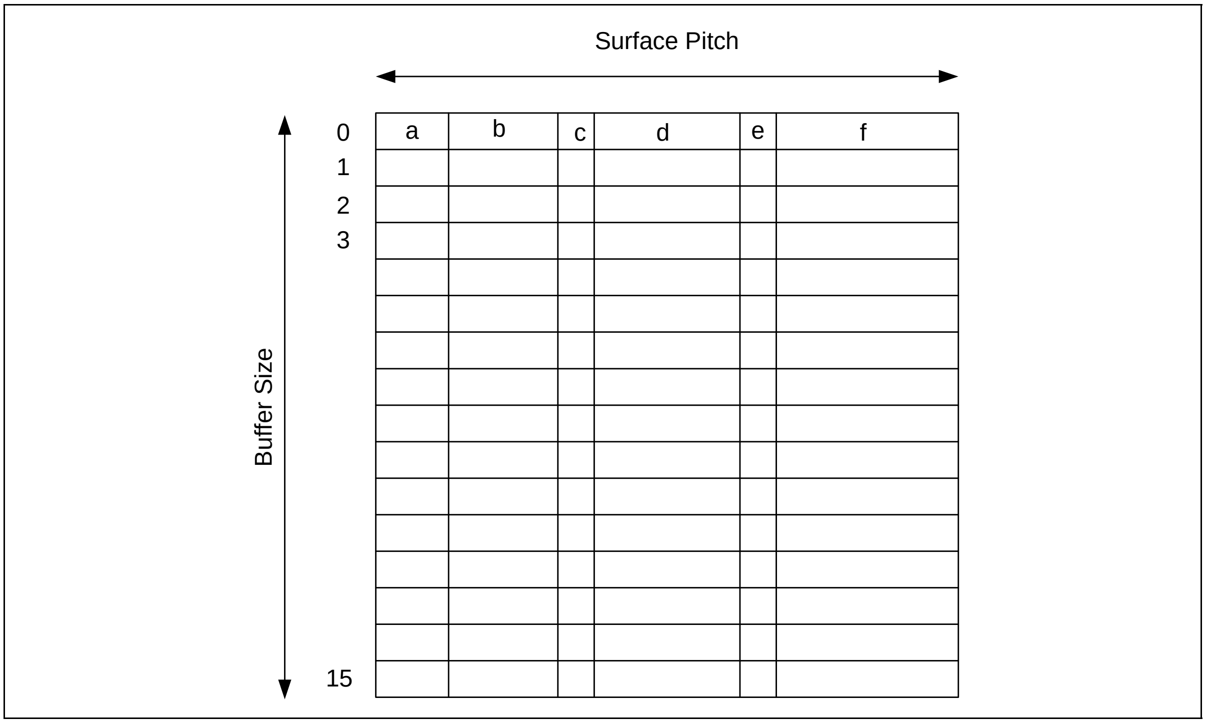
6.17 Surface Layout

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

6.17.1 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.



6.17.2 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.

6.17.3 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).

6.17.3.1 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:

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

6.17.3.2 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:

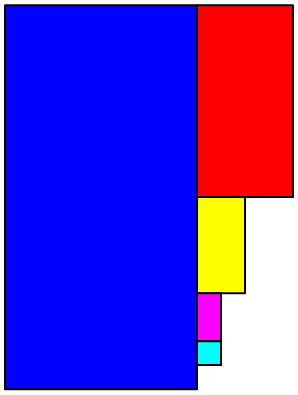
$$w_L = i * ceil\left(\frac{W_L}{i}\right)$$

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

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

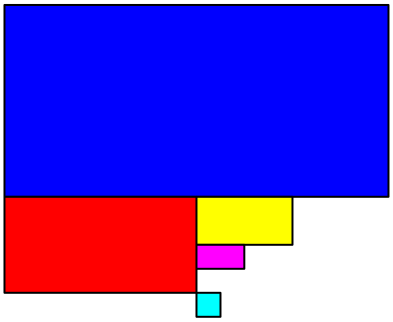
for *MIPLAYOUT_RIGHT* maps:

$$
\begin{aligned}
LOD_0 &= (0,0) \\
LOD_1 &= (w_0, 0) \\
LOD_2 &= (w_0, h_1) \\
LOD_3 &= (w_0, h_1 + h_2) \\
LOD_4 &= (w_0, h_1 + h_2 + h_3) \\
&\ldots
\end{aligned}
$$



for *MIPLAYOUT_BELOW* maps:

$$
\begin{aligned}
LOD_0 &= (0,0) \\
LOD_1 &= (0, h_0) \\
LOD_2 &= (w_1, h_0) \\
LOD_3 &= (w_1, h_0 + h_2) \\
LOD_4 &= (w_1, h_0 + h_2 + h_3) \\
&\ldots
\end{aligned}
$$



6.17.3.3 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.

6.17.3.4 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 6-24. Alignment Units for Texture Maps

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

6.17.3.5 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 BC1, 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 BC2-3 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).

6.17.3.6 Compressed Mipmap Layout

Mipmaps of textures using compressed (BCn, 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.

6.17.3.7 Surface Arrays

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 size for one slice in the texture array. This *QPitch* is multiplied by the array index to and added to the surface base address to determine the base address for that slice. Within the slice, the map is stored identically to a MIPLAYOUT_BELOW 2D surface. *MIPLAYOUT_BELOW is the only format supported by 1D non-arrays and both 2D and 1D arrays, the programming of the MIP Map Layout Mode state variable is ignored when using a TextureArray.*

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

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

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

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

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

6.17.4 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.

6.17.4.1 Hardware Cube Map Layout

The cube face textures are stored in the same way as 3D surfaces are stored (see section 6.17.5 for details). For cube surfaces, however, the depth is equal to the number of faces (always 6) and is not reduced for each MIP. The equation for D_L is replaced with the following for cube surfaces:

$$D_L = 6$$

The "q" coordinate is replaced with the face identifier as follows:

"q" coordinate	face
0	+x
1	-x

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

6.17.4.2 Restrictions

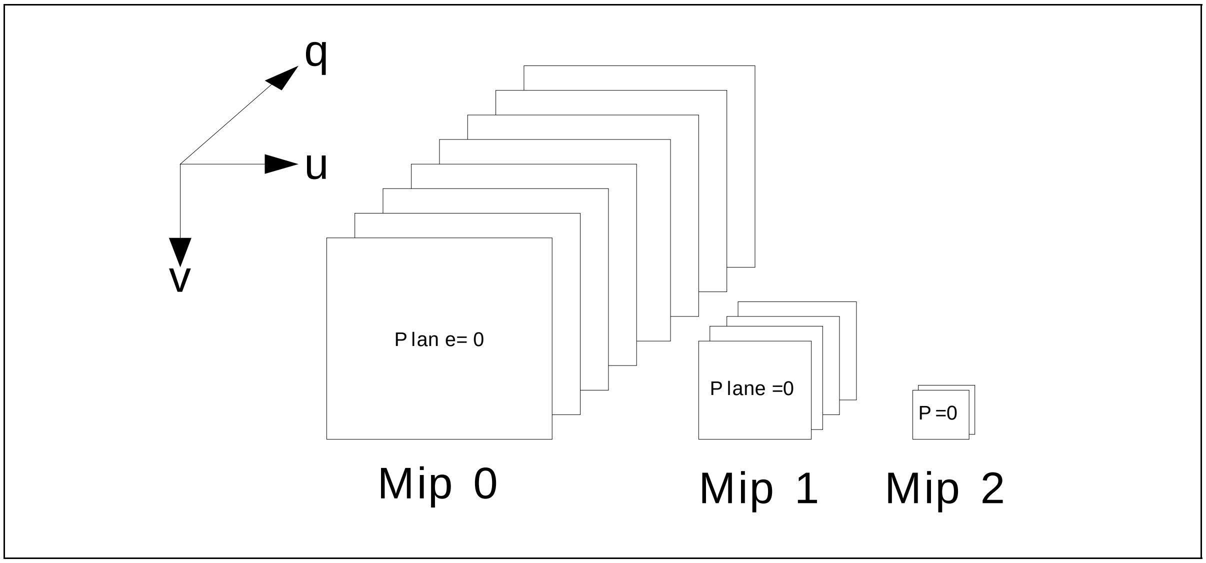
The cube map memory layout is the same whether or not the cube map is mip-mapped, and whether or not all six faces are “enabled”, though the memory backing disabled faces or non-supplied levels can be used by software for other purposes.

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

6.17.5 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 6-5. 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:

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

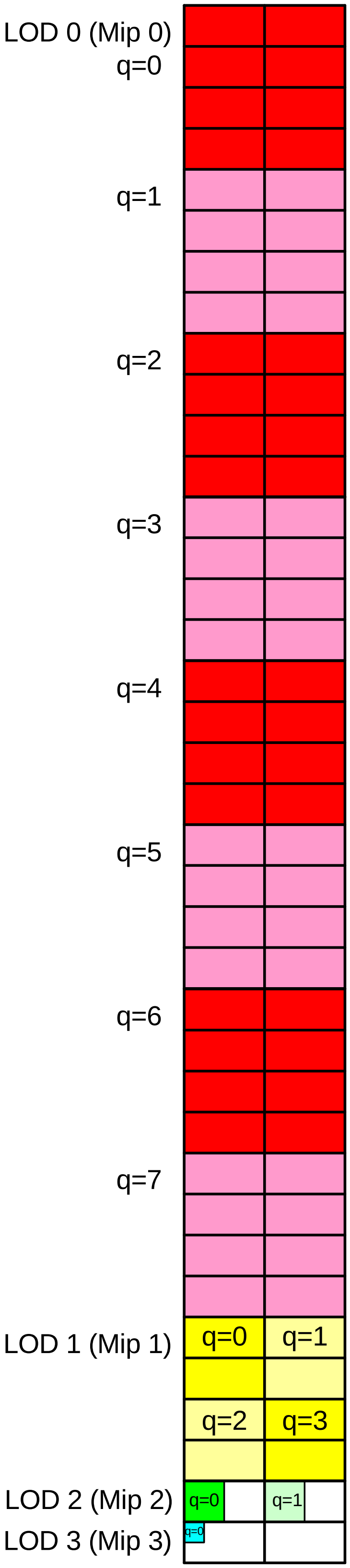
Cache-line aligned width and height are as follows, with I and j being a function of the map format.

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

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

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.



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

...

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.

6.17.5.1 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.

6.18 Surface Padding Requirements

6.18.1 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 6.17.3.4 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.

6.18.2 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.

6.19 Logical Context Data

Logical Contexts are memory images used to store copies of the device’s rendering and ring context.

Logical Contexts are aligned to 256-byte boundaries.

Logical contexts are referenced by their memory address. The format and contents of rendering contexts are considered ***device-dependent*** and software must not access the memory contents directly. The definition of the logical rendering and power context memory formats is included here primarily for internal documentation purposes.

6.19.1 Overall Context Layout

6.19.1.1 Per-Process GTT and Run Lists Disabled

For this case (which is the only case for [DevBW] and [DevCL]), the entire context image consists of the *Register/State Context*, including the pipelined state section.

6.19.1.2 Per-Process GTT and Run Lists Enabled [DevCTG, DevEL]

When a context switch occurs, the head pointer for the current context is first saved back to the head pointer slot of its context descriptor. Then the head and tail pointer offsets and PD information are loaded from the new context descriptor. The **Starting Address** and **Buffer Length** fields of RINGBUF are effectively initialized to LRCA + 4KB and 4 pages, respectively. Command fetch and execution then begins from the head pointer offset into the ring buffer space which begins at LRCA + 4KB. Besides the head pointer, no other context saved and none is restored on BSD context switches.

Register/State Context		
Ring Buffer (4 Pages, 16KB)		
DWord	**Bit**	**Description**
Per-Process HW Status Page		

6.19.2 Register/State Context

DWord	Bit	Description

The following table describes the *device-dependent* layout of a logical context in memory.

Table 6-25 Device-dependemt Layout of a Logical Context

DWord	Bits	State Field
		MEMORY INTERFACE STATE
00h	31:0	**MI_Noop**
01h	31:29	**Instruction Type =** MI_INSTRUCTION = 0h
	28:23	**MI Instruction Opcode =** MI_LOAD_REGISTER_IMM = 22h
	22:12	Reserved: MBZ
	11:8	**Byte Write Disables**: This field specifies which bytes of the **Data DWord** are **not** to be written to the DWord offset specified in *Register Offset*. Format = Enable[4] (bit 8 corresponds to Data DWord [7:0]). Range = Must specify a valid register write operation *This field will always be written as Fh on context saves.*
	7:6	Reserved: MBZ
	5:0	**DWord Length** (Excludes DWord 0,1) = 2bh (dec_44)
02h	31:0	**CACHE_MODE_0 Address**
03h	31:0	**CACHE_MODE_0 Data**
04h	31:0	**CACHE_MODE_1 Address**
05h	31:0	**CACHE_MODE_1 Data**
06h	31:0	**MI_ARB_STATE Address**
07h	31:0	**MI_ARB_STATE Data**
08h	31:0	**INSTPM Address**
09h	31:0	**INSTPM Data**
0Ah	31:0	**IA_VERTICES_COUNT Lower Address**
0Bh	31:0	**IA_VERTICES_COUNT Lower Data**
0Ch	31:0	**IA_VERTICES_COUNT Upper Address**
0Dh	31:0	**IA_VERTICES_COUNT Upper Data**
0Eh	31:0	**IA_PRIMITIVES_COUNT Lower Address**
0Fh	31:0	**IA_PRIMITIVES_COUNT Lower Data**
10h	31:0	**IA_PRIMITIVES_COUNT Upper Address**
11h	31:0	**IA_PRIMITIVES_COUNT Upper Data**
12h	31:0	**VS_INVOCATION_COUNT Lower Address**
13h	31:0	**VS_INVOCATION_COUNT Lower Data**
14h	31:0	**VS_INVOCATION_COUNT Upper Address**
15h	31:0	**VS_INVOCATION_COUNT Upper Data**
16h	31:0	**GS_INVOCATION_COUNT Lower Address**

DWord	Bits	State Field
17h	31:0	**GS_INVOCATION_COUNT Lower Data**
18h	31:0	**GS_INVOCATION_COUNT Upper Address**
19h	31:0	**GS_INVOCATION_COUNT Upper Data**
1ah	31:0	**GS_PRIMITIVES_COUNT Lower Address**
1bh	31:0	**GS_PRIMITIVES_COUNT Lower Data**
1ch	31:0	**GS_PRIMITIVES_COUNT Upper Address**
1dh	31:0	**GS_PRIMITIVES_COUNT Upper Data**
1eh	31:0	**CL_INVOCATION_COUNT Lower Address**
1fh	31:0	**CL_INVOCATION_COUNT Lower Data**
20h	31:0	**CL_INVOCATION_COUNT Upper Address**
21h	31:0	**CL_INVOCATION_COUNT Upper Data**
22h	31:0	**CL_PRIMITIVES_COUNT Lower Address**
23h	31:0	**CL_PRIMITIVES_COUNT Lower Data**
24h	31:0	**CL_PRIMITIVES_COUNT Upper Address**
25h	31:0	**CL_PRIMITIVES_COUNT Upper Data**
26h	31:0	**PS_INVOCATION_COUNT Lower Address**
27h	31:0	**PS_INVOCATION_COUNT Lower Data**
28h	31:0	**PS_INVOCATION_COUNT Upper Address**
29h	31:0	**PS_INVOCATION_COUNT Upper Data**
2Ah	31:0	**PS_DEPTH_COUNT Lower Address**
2Bh	31:0	**PS_DEPTH_COUNT Lower Data**
2Ch	31:0	**PS_DEPTH_COUNT Upper Address**
2Dh	31:0	**PS_DEPTH_COUNT Upper Data**
2Eh	31:0	**MI_Noop**
2Fh	31:0	**MI_Noop**
		PIPELINE_SELECT
30h	31:29	**Instruction Type =** GFXPIPE = 3h
	28:23	**3D Instruction Opcode =** PIPELINE_SELECT GFXPIPE[28:27 = 0h, 26:24 = 1h, 23:16 = 04h] (Non-pipelined)
	22:1	Reserved: MBZ
	0	0: 3D pipeline is selected 1: Media pipeline is selected
		CS_URB_STATE
31h	31:29	**Command Type** = GFXPIPE = 3h
	28:16	**3D Command Opcode** = CS_URB_STATE GFXPIPE[28:27 = 0h, 26:24 = 0h, 23:16 = 01h] (Pipelined)
	15:8	Reserved : MBZ
	7:0	**DWord Length** (excludes DWords 0,1) = 0
32h	31:9	Reserved : MBZ
	8:4	**URB Entry Allocation Size**
	3	Reserved: MBZ
	2:0	**Number of URB Entries**
		URB_FENCE
33h	31:29	**Instruction Type =** GFXPIPE = 3h
	28:16	**3D Instruction Opcode =** URB_FENCE GFXPIPE[28:27 = 0h, 26:24 = 0h, 23:16 = 00h] (Pipelined)
	15:14	Reserved : MBZ
	13	**ModifyEnable(CS Fence)**

DWord	Bits	State Field
	12	**ModifyEnable(VFE Fence)**
	11	**ModifyEnable(SF Fence)**
	10	**ModifyEnable(CLIP Fence)**
	9	**ModifyEnable(GS Fence)**
	8	**ModifyEnable(VS Fence)**
	7:0	**DWord Length** (Excludes DWords 0,1) = 1
34h	31:30	Reserved : MBZ
	29:20	**CLP Fence**
	19:10	**GS Fence**
	9:0	**VS Fence**
35h	31:30	Reserved : MBZ
	29:20	**CS Fence**
	19:10	**VFE Fence**
	9:0	**SF Fence**
CONSTANT_BUFFER		
36h	31:29	**Command Type** = GFXPIPE = 3h
	28:16	**3D Command Opcode** = CONSTANT_BUFFER GFXPIPE[28:27 = 0h, 26:24 = 0h, 23:16 = 02h] (Pipelined)
	15:9	Reserved : MBZ
	8	**Valid** (Saved as *clear* since CONSTANT_BUFFER is saved later)
	7:0	**DWord Length** (excludes DWords 0,1) = 0
37h	31:6	**Buffer Starting Address**
	5:0	**Buffer Length**
STATE_BASE_ADDRESS		
38h	31:29	**Command Type =** GFXPIPE = 3h
	28:16	**3D Command Opcode =** STATE_BASE_ADDRESS GFXPIPE[28:27 = 0h, 26:24 = 1h, 23:16 = 01h] (Nonpipelined)
	15:8	Reserved : MBZ
	7:0	**DWord Length** (Excludes DWords 0,1) = 4
39h	31:12	**General State Base Address**
	11:1	Reserved : MBZ
	0	Modify Enable
3Ah	31:12	**Surface State Base Address**
	11:1	Reserved : MBZ
	0	Modify Enable
3Bh	31:12	**Indirect Object Base Address**
	11:1	Reserved : MBZ
	0	Modify Enable
3Ch	31:12	**General State Access Upper Bound**

DWord	Bits	State Field
	11:1	Reserved : MBZ
	0	Modify Enable
3Dh	31:12	**Indirect Object Access Upper Bound**
	11:1	Reserved: MBZ
	0	Modify Enable
		STATE_SIP
3Eh	31:29	**Command Type =** GFXPIPE = 3h
	28:16	**Command Opcode =** STATE_SIP GFXPIPE[28:27 = 0h, 26:24 = 1h, 23:16 = 02h] (Non-Pipelined)
	15:8	Reserved : MBZ
	7:0	**Vord Length** (Excludes DWords 0,1) = 0
3Fh	31:4	**System Instruction Pointer (SIP)**
	3:0	Reserved : MBZ
		3DSTATE_DRAWING_RECTANGLE
40h	31:29	**Instruction Type =** GFXPIPE = 3h
	28:16	**3D Instruction Opcode =** 3DSTATE_DRAWING_RECTANGLE GFXPIPE[28:27 = 3h, 26:24 = 1h, 23:16 = 00h] (Non-Pipelined)
	15:0	ngth **(Excludes DWord 0,1) = 2**
41h	31:16	**Clipped Drawing Rectangle Y Min**
	15:0	**Clipped Drawing Rectangle X Min**
42h	31:16	**Clipped Drawing Rectangle Y Max**
	15:0	**Clipped Drawing Rectangle X Max**
43h	31:16	**Drawing Rectangle Origin Y**
	15:0	**Drawing Rectangle Origin X**
		3DSTATE_DEPTH_BUFFER
44h	31:29	**Instruction Type =** GFXPIPE = 3h
	28:16	**3D Instruction Opcode =** 3DSTATE_DEPTH_BUFFER GFXPIPE[28:27 = 3h, 26:24 = 1h, 23:16 = 05h] (Non-Pipelined)
	15:0	**DWord Length** (excl. DWord 0,1) = 3
45h	31:29	**Surface Type**
	28	**Reserved: MBZ**
	27	**Tiled Surface**
	26	**Tile Walk**
	25	**Depth Buffer Coordinate Offset Disable**
	24:21	Reserved : MBZ
	20:18	**Surface Format**
	17:0	**Surface Pitch**
46h	31:0	**Surface Base Address**
47h	31:19	**Height**
	18:6	**Width**
	5:2	**LOD**
	1	**MIP Map Layout Mode**
	0	Reserved : MBZ
48h	31:21	**Depth**
	20:12	**Minimum Array Element**
	11:0	Reserved : MBZ
		3DSTATE_CHROMA_KEY (INDEX_0)
49h	31:29	**Instruction Type =** 3D_INSTRUCTION = 3h
	28:16	**3D Instruction Opcode =** 3DSTATE_CHROMA_KEY GFXPIPE[28:27 = 3h, 26:24 = 1h; 23:16 = 04h] (Non-Pipelined)

DWord	Bits	State Field
	15:0	**DWord Length** (Excludes DWords 0,1) = 2
4Ah	31:30	**ChromaKey Table Index = 0**
	29:0	Reserved: MBZ
4Bh	31:0	**ChromaKey Low Value**
4Ch	31:0	**ChromaKey High Value**
3DSTATE_CHROMA_KEY (INDEX_1)		
4Dh	31:29	**Instruction Type =** 3D_INSTRUCTION = 3h
	28:16	**3D Instruction Opcode =** 3DSTATE_CHROMA_KEY GFXPIPE[28:27 = 3h, 26:24 = 1h; 23:16 = 04h] (Non-Pipelined)
	15:0	**DWord Length** (Excludes DWords 0,1) = 2
4Eh	31:30	**ChromaKey Table Index = 1**
	29:0	Reserved: MBZ
4Fh	31:0	**ChromaKey Low Value**
50h	31:0	**ChromaKey High Value**
3DSTATE_CHROMA_KEY (INDEX_2)		
51h	31:29	**Instruction Type =** 3D_INSTRUCTION = 3h
	28:16	**3D Instruction Opcode =** 3DSTATE_CHROMA_KEY GFXPIPE[28:27 = 3h, 26:24 = 1h; 23:16 = 04h] (Non-Pipelined)
	15:0	**DWord Length** (Excludes DWords 0,1) = 2
52h	31:30	**ChromaKey Table Index = 2**
	29:0	Reserved: MBZ
53h	31:0	**ChromaKey Low Value**
54h	31:0	**ChromaKey High Value**
3DSTATE_CHROMA_KEY (INDEX_3)		
55h	31:29	**Instruction Type =** 3D_INSTRUCTION = 3h
	28:16	**3D Instruction Opcode =** 3DSTATE_CHROMA_KEY GFXPIPE[28:27 = 3h, 26:24 = 1h; 23:16 = 04h] (Non-Pipelined)
	15:0	**DWord Length** (Excludes DWords 0,1) = 2
56h	31:30	**ChromaKey Table Index = 3**
	29:0	Reserved: MBZ
57h	31:0	**ChromaKey Low Value**
58h	31:0	**ChromaKey High Value**
3D State Constant Color		
59h	31:29	**Instruction Type =** GFXPIPE = 3h
	28:16	**3D Instruction Opcode =** 3DSTATE_CONSTANT_COLOR GFXPIPE[28:27 = 3h, 26:24 = 1h, 23:16 = 01h] (Non-Pipelined)
	15:0	**DWord Length** (excl. DWord 0,1) = 3
5Ah	31:0	**Blend Constant Color Red**
5Bh	31:0	**Blend Constant Color Blue**
5Ch	31:0	**Blend Constant Color Green**
5Dh	31:0	**Blend Constant Color Alpha**
3DSTATE_GLOBAL_DEPTH_OFFSET_CLAMP		
5Eh	31:29	**Instruction Type =** 3D_INSTRUCTION = 3h
	28:16	**3D Instruction Opcode =** 3DSTATE_GLOABL_DEPTH_OFFSET_CLAMP GFXPIPE[28:27 = 3h, 26:24 = 1h, 23:16 = 09h] (Non-Pipelined)
	15:0	**DWord Length** (excl. DWord 0,1) = 0
5Fh	31:0	**Global Depth Offset Clamp**
3DSTATE_POLY_STIPPLE_OFFSET		
60h	31:29	**Instruction Type =** 3D_INSTRUCTION = 3h

DWord	Bits	State Field
	28:16	**3D Instruction Opcode =** 3DSTATE_POLY_STIPPLE_OFFSET GFXPIPE[28:27 = 3h, 26:24 = 1h, 23:16 = 06h] (Non-Pipelined)
	15:0	**DWord Length** (excl. DWord 0,1) = 0
61h	31:13	Reserved: MBZ
	12:8	**Polygon Stipple X Offset**
	7:5	Reserved: MBZ
	4:0	**Polygon Stipple Y Offset**
3DSTATE_LINE_STIPPLE		
62h	31:29	**Instruction Type =** 3D_INSTRUCTION = 3h
	28:16	**3D Instruction Opcode =** 3DSTATE_LINE_STIPPLE GFXPIPE[28:27 = 3h, 26:24 = 1h, 23:16 = 08h] (Non-Pipelined)
	15:0	**DWord Length** (excl. DWord 0,1) = 1
63h	31	Modify Enable (**Current Repeat Counter**, **Current Stipple Index**)
	30	Reserved: MBZ
	29:21	**Current Repeat Counter** This field sets the HW-internal repeat counter state. Format = U9
	20	Reserved: MBZ
	19:16	**Current Stipple Index** This field sets the HW-internal stipple pattern index. Format = U4
	15:0	**Line Stipple Pattern** Specifies a pattern used to mask out bit specific pixels while rendering lines. Format = 16 bit mask. Bit 15 = most significant bit, Bit 0 = least significant bit
64h	31:16	**Line Stipple Inverse Repeat Count**
	15:9	Reserved: MBZ
	8:0	**Line Stipple Repeat Count**
SVGunit Context Data (Media)		
MEDIA_STATE_POINTERS **Note: Dwords 65h – 67h will be saved as MI_NOOP (opcode 00h) unless MEDIA_STATE_POINTERS has been initialized (issued at least once).**		
65h	31:29	**Command Type =** GFXPIPE = 3h
	28:16	**Media Command Opcode =** MEDIA_STATE_POINTERS Pipeline[28:27] = Media = 2h; Opcode[26:24] = 0h; Subopcode[23:16] **=** 0h
	15:0	**DWord Length** (Excludes DWords 0,1) = 01h
66h	31:5	**Pointer to VLD_STATE**
	4:1	Reserved : MBZ
	0	**VLD Enable**
67h	31:5	**Pointer to VFE_STATE**
	4:0	Reserved : MBZ
SVGunit Context Data (3D)		
3DSTATE_PIPELINE_POINTERS **Note: Dwords 68h – 6Eh will be saved as MI_NOOP (opcode 00h) unless 3DSTATE_PIPELINE_POINTERS has been initialized (issued at least once).**		
68h	31:29	**Command Type =** GFXPIPE = 3h
	28:16	**3D Command Opcode =** 3DSTATE_PIPELINED_POINTERS GFXPIPE[28:27 = 3h, 26:24 = 0h, 23:16 = 00h] (Pipelined)
	15:8	Reserved : MBZ
	7:0	**DWord Length** (Excludes DWords 0,1) = 5
69h	31:5	**Pointer to VS_STATE**
	4:0	Reserved : MBZ
6Ah	31:5	**Pointer to GS_STATE**
	4:1	Reserved : MBZ
	0	**GS Enable**

DWord	Bits	State Field
6Bh	31:5	**Pointer to CLP_STATE**
	4:1	Reserved : MBZ
	0	**CLP Enable**
6Ch	31:5	**Pointer to SF_STATE**
	4:0	Reserved : MBZ
6Dh	31:5	**Pointer to WINDOWER_STATE**
	4:0	Reserved : MBZ
6Eh	31:6	**Pointer to COLOR_CALC_STATE**
	5:0	Reserved : MBZ
3DSTATE_BINDING_TABLE_POINTERS **Note: Dwords 6Fh – 74h will be saved as MI_NOOP (opcode 00h) unless 3DSTATE_BINDING_TABLE_POINTERS has been initialized (issued at least once).**		
6Fh	31:29	**Command Type =** GFXPIPE = 3h
	28:16	**3D Command Opcode =** 3DSTATE_BINDING_TABLE_POINTERS GFXPIPE[28:27 = 3h, 26:24 = 0h, 23:16 = 01h] (Pipelined)
	15:8	Reserved : MBZ
	7:0	**DWord Length** (Excludes DWords 0,1) = 4
70h	31:5	**Pointer to VS Binding Table**
	4:0	Reserved : MBZ
71h	31:5	**Pointer to GS Binding Table**
	4:0	Reserved : MBZ
72h	31:5	**Pointer to CLP Binding Table**
	4:0	Reserved : MBZ
73h	31:5	**Pointer to SF Binding Table**
	4:0	Reserved : MBZ
74h	31:5	**Pointer to PS Binding Table**
	4:0	Reserved : MBZ
CONSTANT_BUFFER **Note: Dwords 75h – 76h will be saved as MI_NOOP (opcode 00h) unless CONSTANT_BUFFER has been initialized (issued at least once).**		
75h	31:29	**Command Type** = GFXPIPE = 3h
	28:16	**3D Command Opcode** = CONSTANT_BUFFER GFXPIPE[28:27 = 0h, 26:24 = 0h, 23:16 = 02h] (Pipelined)
	15:9	Reserved : MBZ
	8	**Valid** (Will be *set* if CONSTANT_BUFFER was issued in the context to be saved)
	7:0	**DWord Length** (excludes DWords 0,1) = 0
76h	31:6	**Buffer Starting Address**
	5:0	**Buffer Length**
77h	31:0	**MI_Noop**
(This region was formerly Blitter Related Context Data)		
78 – 87h	31:0	**Reserved** Should be treated as garbage data when inspecting a saved context.
VFunit Related Context Data		
3DSTATE_INDEX_BUFFER		
88h	31:29	**Command Type =** GFXPIPE = 3h
	28:16	**GFXPIPE Opcode =** 3DSTATE_INDEX_BUFFER GFXPIPE[28:27 = 3h, 26:24 = 0h, 23:16 = 0Ah] (Pipelined)
	15:11	Reserved : MBZ
	10	**Cut Index Enable**
	9:8	**Index Format**
	7:0	**DWord Length** (excludes DWords 0,1) = 1
89h	31:0	**Buffer Starting Address**
8Ah	31:0	**Buffer Ending Address**

DWord	Bits	State Field
3DSTATE_VERTEX_BUFFER		
8Bh	31:29	**Command Type =** GFXPIPE = 3h
	28:16	**GFXPIPE Opcode =** 3DSTATE_VERTEX_BUFFERS GFXPIPE[28:27 = 3h, 26:24 = 0h, 23:16 = 08h] (Pipelined)
	15:8	Reserved : MBZ
	7:0	**DWord Length** (excludes DWords 0,1)
8C-8Fh		**Vertex Buffer 0 State**
90-93		**Vertex Buffer 1 State**
		…
CC-CFh		**Vertex Buffer 16 State**
3DSTATE_VERTEX_ELEMENT (71 - 93h)		
D0h	31:29	**Command Type =** GFXPIPE = 3h
	28:16	**GFXPIPE Opcode =** 3DSTATE_VERTEX_ELEMENTS GFXPIPE[28:27 = 3h, 26:24 = 0h, 23:16 = 09h] (Pipelined)
	15:8	Reserved : MBZ
	7:0	**DWord Length** (excludes DWords 0,1)
D1 – D2h	[1-2] dw	**Element[0]**
D3 – D4h	[3-4] dw	**Element[1]**
…	…	…
F3 – F4h	[37-38]dw	**Element[17]**
3DSTATE_VERTEX_STATISTIC_Counter_ENABLE		
F5h	31:29	**Command Type =** GFXPIPE = 3h
	28:16	**GFXPIPE Opcode =** 3DSTATE_VF_STATISTICS GFXPIPE[28:27 = 3h, 26:24 = 0h, 23:16 = 0Bh] (Pipelined)
	15:1	Reserved : MBZ
	0	**Statistics Enable**
F6-FFh		**MI_NOOP**
DMunit Related Context Data		
3DSTATE_SAMPLER_PALETTE_LOAD (ONLY on Extended SAVE Mode)		
100h	31:29	**Instruction Type =** GFXPIPE = 3h
	28:16	**3D Instruction Opcode =** 3DSTATE_SAMPLER_PALETTE_LOAD GFXPIPE[28:27 = 3h, 26:24 = 1h; 23:16 = 02h] (Non-Pipelined)
	15:4	Reserved: MBZ
	3:0	**DWord Length** (excludes DWords 0,1)
101-110h	31:24	Reserved
	23:0	**Palette Color[0:N-1]**
111-117h	31:0	**MI_NOOP**
WIZunit Related Context Data		
3DSTATE_POLY_STIPPLE_PATTERN (ONLY on Extended SAVE Mode)		
118h	31:29	**Instruction Type =** 3D_INSTRUCTION = 3h
	28:16	**3D Instruction Opcode =** 3DSTATE_POLY_STIPPLE_PATTERN GFXPIPE[28:27 = 3h, 26:24 = 1h, 23:16 = 07h] (Non-Pipelined)
	15:0	**DWord Length** (excl. DWord 0,1) = 31
119h	31:0	**Polygon Stipple Pattern Row 1** (top most)
11Ah	31:0	**Polygon Stipple Pattern Row 2**
11Bh – 138h	31:0	**Polygon Stipple Pattern Rows 3** through **32** (bottom-most)

DWord	Bits	State Field
139-13Fh	31:0	**MI_Noop**

6.19.2.1.1 Power Context Memory Layout ([DevCL] Only)

Additional context data is required if a reset occurs (if power is lost, for example) between the save and restore of a context. A mobile-only feature provides for saving and restoring the following context state/registers in this event. Note that the context below includes a pointer (in an MI_SET_CONTEXTcommand) to the usual logical rendering context which is considered a subset of the power context when power context is saved/restored. See the device EDS for further details.

DWord	Bits	State Field
		MEMORY INTERFACE STATE
00h	31:0	MI_NOOP
01h	31:29	**Instruction Type** = MI_INSTRUCTION = 0h
	28:23	**MI Instruction Opcode** = MI_LOAD_REGISTER_IMM = 22h
	22:12	Reserved: MBZ
	11:8	**Byte Write Disables** = Fh (all enabled)
	7:6	Reserved: MBZ
	5:0	**DWord Length** (Excludes DWord 0,1) = Ah
02h	31:0	**Scratch Pad Register Address Offset**
03h	31:0	**Scratch Pad Register Data**
04h	31:0	EXCC Register Address Offset
05h	31:21	Reserved. MBZ.
	20:16	**Bit Write Masks for Bits 4:0: Written as 1Fh (all enabled)**
	15:5	Reserved: MBZ
	4:0	**User Defined Condition Codes**
06h	31:0	**Ring Buffer Tail Pointer Register Offset**
07h	31:21	Reserved: MBZ
	20:3	**Tail Offset (Never Saved on Context Switch)**
	2:1	Reserved: MBZ
	0	**In Use (Always saved as 0)**
08h	31:0	**Ring Buffer Starting Address Register Offset**
09h	31:12	**Starting Address**
	11:0	Reserved: MBZ
0Ah	31:0	**Ring Buffer Head Pointer Register Offset** *Note: The Head reg is restored after the Address reg, as restoring the Address reg resets the Head.*
0Bh	31:21	**Wrap Count**

DWord	Bits	State Field
	20:2	**Head Offset**
	1:0	Reserved: MBZ
0Ch	31:0	**Ring Buffer Length Register Offset**
0Dh	31:21	Reserved: MBZ
	20:12	**Buffer Length**
	11	**RB Wait**
	10	**RB Arb off**
	9	**RB in time slice**
	8	**Disable Register Accesses**
	7:3	Reserved: MBZ
	2:1	**Automatic Report Head Pointer**
	0	**Ring Buffer Enable**
0Eh	31:29	**Instruction Type =** MI_INSTRUCTION = 0h
	28:23	**MI Instruction Opcode =** MI_SET_CONTEXT = 18h
	22:6	Reserved: MBZ
	5:0	**DWord Length** (Excludes Dword 0,1) = 0
0Fh	31:11	**Logical Context Address**
	10:4	Reserved: MBZ
	8	**Memory Space Select**
	7:4	**Physical Start Address Extension**
	3	**Extended State Save Enable**
	2	**Extended State Restore Enable**
	1	**Force Restore**
	0	**Restore Inhibit**

6.19.2.1.2 Logical Context Initialization

Each logical context should initialize all device state before beginning operations so that any context switches that occur subsequently will save and restore coherent device state. See *Memory Interface Functions* for more information. The following table provides values that should be used to initialize any state that the context does not require for its operations. Note that these state variables will need to be set to something more intelligent for a context that intends to perform operations that depend on them. The values of these state variables are saved (and subsequently restored) on any context switch, with the exception of the 3DSTATE_SAMPLER_PALETTE_LOAD and 3DSTATE_POLY_STIPPLE_PATTERN which are only saved from and restored to contexts that have the **Extended State Save Enable** and **Extended State Restore Enable**, respectively, set in the MI_SET_CONTEXT command that triggers the context switch. See *Memory Interface Commands* for details of this command.

Note that 3D/Media *pipelined* state cannot be initialized; it is not stored internally to the device but is accessed from state blocks in memory as required by rendering operations. Any context that will issue 3DPRIMITIVE or MEDIA_OBJECT_LOAD commands must first place valid state structures in memory and send down the corresponding command (3DSTATE_PIPELINED_POINTERS or MEDIA_STATE_POINTERS) to point to it. There are no defaults for this state. The following table (Table 6-26) summarizes state that MUST BE properly set up for a given context. Please refer to the *Graphics Processing Engine* (GPE), *3D Pipeline* and *Media* chapters for details on these commands.

Table 6-26. Context Setup that Cannot Use Defaults

Context	Required Setup	Notes
3D	**PIPELINE_SELECT**	**3D Pipeline must be selected**
	CS_URB_STATE	**Must allocate sufficient URB space for constants that will be used.**
	3DSTATE_PIPELINED_POINTERS	**Pointers for all enabled FF units (when offset from base address) must point to valid state in memory.**
	3DSTATE_BINDING_TABLE_POINTERS	**Pointers for all enabled FF units (when offset from base address) must point to valid binding tables in memory.**
	STATE_BASE_ADDRESS	**Must be properly initialized so that pointers above point to valid state blocks.**
	URB_FENCE	**Enabled FF units must be allocated sufficient URB space to avoid deadlock. Note that most FF units *cannot* be disabled. Only VS and CLIP can be disabled.**
	CONSTANT_BUFFER	**Must point to a valid constant buffer if constants will be used.**
	STATE_SIP	**Must point to a valid exception handler if any threads will be dispatched with any exceptions enabled.**

Context	Required Setup	Notes
Media	**PIPELINE_SELECT**	**Media Pipeline must be selected**
	CS_URB_STATE	**Same as above**
	MEDIA_STATE_POINTERS	**Pointers for one, or both if enabled, Media FF units (when offset from base address) must point to valid state in memory.**
	STATE_BASE_ADDRESS	**Must be properly initialized so that pointers above point to valid state blocks.**
	URB_FENCE	**Enabled FF units must be allocated sufficient URB space to avoid deadlock. Note that the VFE FF unit *cannot* be disabled.**
	CONSTANT_BUFFER	**Must point to a valid constant buffer if constants will be used.**
	STATE_SIP	**Must point to a valid exception handler if any threads will be dispatched with any exceptions enabled.**

Table 6-27. Initialization of Command State

Instruction/Field	Value
PIPELINE_SELECT	
Pipeline Select	0 = 3D pipeline is selected
CS_URB_STATE	
URB Entry Allocation Size	0
Number of URB Entries	0
URB_FENCE	
CS Unit URB Reallocation Request	0
VFE Fence Unit URB Reallocation Request	0
SF Unit URB Reallocation Request	0
CLIP Unit URB Reallocation Request	0
GS Unit URB Reallocation Request	0
VS Unit URB Reallocation Request	0
CLP Fence	192
GS Fence	128
VS Fence	64
CS Fence	256
VFE Fence	0
SF Fence	252
CONSTANT_BUFFER	
Valid	0
Buffer Starting Address	0
Buffer Length	0

Instruction/Field	Value
STATE_BASE_ADDRESS	
General State Base Address	0
Surface State Base Address	0
Indirect Object Base Address	0
General State Access Upper Bound	0
Indirect Object Access Upper Bound	0
STATE_SIP	
System Instruction Pointer	0
3DSTATE_DRAWING_RECTANGLE	
Clipped Drawing Rectangle Y Min	0
Clipped Drawing Rectangle X Min	0
Clipped Drawing Rectangle Y Max	8191
Clipped Drawing Rectangle X Max	8191
Drawing Rectangle Origin Y	0
Drawing Rectangle Origin X	0
3DSTATE_DEPTH_BUFFER	
Surface Type	7 (SURFTYPE_NULL)
Tiled Surface	0
Tile Walk	1 = Y
Depth Buffer Coordinate Offset Disable	0
Surface Format	0
Surface Pitch	0
Surface Base Address	0
Height	0
Width	0
LOD	0
MIP Map Layout Mode	0 = MIPLAYOUT_BELOW
Depth	0
Minimum Array Element	0
3DSTATE_CHROMA_KEY (INDEX_0)	
ChromaKey Table Index	0
ChromaKey Low Value	0
ChromaKey High Value	0
3DSTATE_CHROMA_KEY (INDEX_1)	
ChromaKey Table Index	1
ChromaKey Low Value	0
ChromaKey High Value	0
3DSTATE_CHROMA_KEY (INDEX_2)	
ChromaKey Table Index	2
ChromaKey Low Value	0
ChromaKey High Value	0
3DSTATE_CHROMA_KEY (INDEX_3)	
ChromaKey Table Index	3
ChromaKey Low Value	0
ChromaKey High Value	0

Instruction/Field	Value
3DSTATE_CONSTANT_COLOR	
Blend Constant Color Red	1.0
Blend Constant Color Blue	1.0
Blend Constant Color Green	1.0
Blend Constant Color Alpha	1.0
3DSTATE_GLOBAL_DEPTH_OFFSET_CLAMP	
Global Depth Offset Clamp	0.0
3DSTATE_POLY_STIPPLE_OFFSET	
Polygon Stipple X Offset	0
Polygon Stipple Y Offset	0
3DSTATE_LINE_STIPPLE	
Modify Enable	0
Current Repeat Counter	0
Current Stipple Index	0
Line Stipple Pattern	0
Line Stipple Inverse Repeat Count	0
Line Stipple Repeat Count	0
MEDIA_STATE_POINTERS	
Pointer to VLD_STATE	0
VLD Enable	0
Pointer to VFE_STATE	0
3DSTATE_PIPELINE_POINTERS	
Pointer to VS_STATE	0
Pointer to GS_STATE	0
GS Enable	0
Pointer to CLP_STATE	0
CLP Enable	0
Pointer to SF_STATE	0
Pointer to WINDOWER_STATE	0
Pointer to COLOR_CALC_STATE	0
3DSTATE_BINDING_TABLE_POINTERS	
Pointer to VS Binding Table	0
Pointer to GS Binding Table	0
Pointer to CLP Binding Table	0
Pointer to SF Binding Table	0
Pointer to PS Binding Table	0
3DSTATE_INDEX_BUFFER	
Cut Index Enable	0
Index Format	0
Buffer Starting Address	0
Buffer Ending Address	0
3DSTATE_VERTEX_BUFFER (0 – 16)	
DWord Length (excludes DWords 0,1)	50 (32h)
Vertex Buffer Index	0
Buffer Access Type	0 = VERTEXDATA

Instruction/Field	Value
Buffer Pitch	0
Buffer Starting Address	0
Max Index	0
… values repeated for all 17 Vertex Buffers	…
3DSTATE_VERTEX_ELEMENT (0 – 17)	
DWord Length (excludes DWords 0,1)	35 (23h)
Vertex Buffer Index	0
Valid	0
Source Element Format	0
Source Element Offset	0
Component 0 Control	2 = VFCOMP_STORE_0
Component 1 Control	0 = VFCOMP_NOSTORE
Component 2 Control	0 = VFCOMP_NOSTORE
Component 3 Control	0 = VFCOMP_NOSTORE
Destination Element Offset	0
… values repeated for all 18 Vertex Elements	…
3DSTATE_VF_STATISTICS	
Statistics Enable	0
3DSTATE_SAMPLER_PALETTE_LOAD (Required to be initialized only if context uses extended save)	
DWord Length (excludes DWords 0,1)	15
Palette Color 0	0
Palette Color 1	0
…	0
Palette Color 15	0
3DSTATE_POLY_STIPPLE_PATTERN (Required to be initialized only if context uses extended save)	
DWord Length (excl. DWord 0,1)	31
Polygon Stipple Pattern Row 1 (top most)	0
Polygon Stipple Pattern Row 2	0
…	0
Polygon Stipple Pattern Row 32 (bottom-most)	0

6.19.2.2 [DevCTG-B], [DevEL]

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-8h	Non-Pipelined 3D State Context Data
9h-19h	Sampler Palette Load (Extended Only)
1Ah-1Ch	Poly Stipple Pattern (Extended Only)
1Dh-1Eh	Reserved
1Fh	Media PRT
20h-27h	Pipelined 3D and Media State (Stored Here Only When PPGTT/Runlists Disabled)
27h-3Fh	Reserved

Ring Registers and Non-Pipelined Context Details:

		Valid Only When Run Lists and PPGTT Enabled					
DW Range	**DW Count**	**State Field**	**Render Restore Inhibited**	**PPGTT and Run Lists Enabled**	**PPGTT and Run Lists 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	NR	S/R	X	X	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
07h	1	PD Base Virtual Address Register	R	R	X	X	Yes
08h	1	Read Offset in Piipelined State Page (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

		Valid Only When Run Lists and PPGTT Enabled					
DW Range	**DW Count**	**State Field**	**Render Restore Inhibited**	**PPGTT and Run Lists Enabled**	**PPGTT and Run Lists Disabled**	**Power Context**	**Set Before Submitting Context?**
0Bh	1	Committed Primitive ID	NR	S/R	X	X	No
0Ch	1	Super Span Count	NR	S/R	X	X	No
0Dh	1	VFE Debug Counter	NR	S/R	X	X	No
0Eh	1	**Reserved(For power context: register 2180)**	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					
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	MI_ARB_STATE Register					
4Dh	1	INSTPM Register					
4Eh	1	EXCC Register					
4Fh	1	MI_MODE Register					
50h	1	PIPELINE_SELECT					
51h – 56h	6	STATE_BASE_ADDRESS					
57h – 58h	2	STATE_SIP					
59h – 5Ch	4	3DSTATE_DRAWING_RECTANGLE					
5Dh – 5Fh	3	3DSTATE_AA_LINE_PARAMS					

		Valid Only When Run Lists and PPGTT Enabled					
DW Range	**DW Count**	**State Field**	**Render Restore Inhibited**	**PPGTT and Run Lists Enabled**	**PPGTT and Run Lists Disabled**	**Power Context**	**Set Before Submitting Context?**
60h – 65h	6	3DSTATE_DEPTH_BUFFER					
66h – 6Ah	5	3DSTATE_CONSTANT_COLOR					
6Bh – 6Ch	2	3DSTATE_POLY_STIPPLE_OFFSET					
6Dh – 6Fh	3	3DSTATE_LINE_STIPPLE					
70h – 71h	2	3DSTATE_GLOBAL_DEPTH_OFFSET_CLAMP					
72h – 7Fh	14	**Reserved**					
80h – 83h	4	3DSTATE_CHROMA_KEY (0)					
84h – 87h	4	3DSTATE_CHROMA_KEY (1)					
88h – 8Bh	4	3DSTATE_CHROMA_KEY (2)					
8Ch – 8Fh	4	3DSTATE_CHROMA_KEY (3)					
90h – 190h	257	3DSTATE_SAMPLER_PALETTE_LOAD_0					
191h – 196h	6	**Reserved**					
197h – 297h	257	3DSTATE_SAMPLER_PALETTE_LOAD_1					
298h – 29Fh	8	**Reserved**					
2A0h – 2C0h	33	3DSTATE_POLY_STIPPLE_PATTERN					
2C1h – 2CFh	15	**Reserved**					
2D0h – 2EFh	32	**Reserved**					
2F0h – 2FFh	16	Media PRT Data					
		The following state is saved here only when Run Lists are not being used. If Run Lists are enabled, this state will be saved in the Pipelined State page. Note that these commands are saved without headers (unlike the commands above).					
300h – 304h	5	3DSTATE_PIPELINED_POINTERS					
305h – 309h	5	3DSTATE_BINDING_TABLE_POINTERS					
30Ah – 30Bh	2	MEDIA_STATE_POINTERS					
30Ch – 30Eh	3	URB_FENCE					
30Fh	1	CS_URB_STATE					
310h – 311h	2	CONSTANT_BUFFER					
312h – 314h	3	3DSTATE_INDEX_BUFFER					
315h – 358h	68	3DSTATE_VERTEX_BUFFERS					
359h – 37Ch	36	3DSTATE_VERTEX_ELEMENTS					
37Dh	1	3DSTATE_VF_STATISTICS					
37Eh – 3FFh	130	**Reserved**					

6.19.3 The Probe List

The Probe List consists of 1024 slots. Each slot can hold a probe list entry. Each entry is one Dword and has the following format:

Bit	Description
31:12	**Surface Page Base Address.** Format = PerProcessGraphicsVirtualAddress[31:12]
11:1	**Reserved.** MBZ
0	**Fault.** This bit is set by HW if this probe faults (either on context restore or when executing MI_PROBE.) This bit is ignored when this probe entry is read in order to be re-checked as part of a context restore operation.

SW must clear the **Fault** bit in a probe list entry for which it has successfully serviced a surface fault. When restoring a context, **Fault** bits are only set for new faults. They are not cleared for reprobes which do not fault.

6.19.4 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.

6.19.5 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.

6.19.6 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:1A	**Reserved.**
19	**Context Save Finished Timestamp**
18	**Context Restore Complete Timestamp**
17	**Pre-empt Request Received Timestamp**
16	**Last Switch Timestamp**
15:12	**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).

§§

7 Device 2 Configuration Registers

7.1 Introduction

PCI Configuration Device 2 is the Internal Graphics Device (IGD). The common subset of these registers is thus documented in this specification. For all other configuration register devices, please see the EDS for the particular device concerned.

Note that only a subset of the Device 2 Configuration registers is documented here. Registers that are not documented here are available for use (and many are already used) for product-specific control registers that relate to Device 2. Please see the EDS for the complete set of Device 2 registers for a given product.

All registers documented herein are common between all products in the GenX family except for the minor exceptions noted. Any changes to the registers documented here must be presented to the common graphics core change control board.

7.2 Device 2, Function 0

Register Name	Register Symbol	Register Start	Register End	Default Value	Access
Vendor Identification	VID2	0	1	8086h	RO;
Device Identification	DID2	2	3	[Device Specific]	RO;
PCI Command	PCICMD2	4	5	0000h	RO; R/W;
PCI Status	PCISTS2	6	7	0090h	RO;
Revision Identification	RID2	8	8	00h	RO;
Class Code	CC	9	B	030000h	RO;
Cache Line Size	CLS	C	C	00h	RO;
Master Latency Timer	MLT2	D	D	00h	RO;
Header Type	HDR2	E	E	80h	RO;
Built In Self Test	BIST	F	F	00h	RO;
Graphics Translation Table Range Address	GTTMMADR	10	17	0000000000000004h	RO; R/W;
Graphics Memory Range Address	GMADR	18	1F	000000000000000Ch	RO; R/W; R/W/L;
I/O Base Address	IOBAR	20	23	00000001h	RO; R/W;
Subsystem Vendor Identification	SVID2	2C	2D	0000h	R/WO;
Subsystem Identification	SID2	2E	2F	0000h	R/WO;

Register Name	Register Symbol	Register Start	Register End	Default Value	Access
Video BIOS ROM Base Address	ROMADR	30	33	00000000h	RO;
Capabilities Pointer	CAPPOINT	34	34	90h	RO;
Interrupt Line	INTRLINE	3C	3C	00h	R/W;
Interrupt Pin	INTRPIN	3D	3D	01h	RO;
Minimum Grant	MINGNT	3E	3E	00h	RO;
Maximum Latency	MAXLAT	3F	3F	00h	RO;
Capabilities Pointer (to Mirror of Dev0 CAPID)	MCAPPTR	44	44	48h	RO;
Mirror of Dev 0 Capability Identification	MCAPID	48	51	[Device Specific]	RO;
Mirror of Dev0 GMCH Graphics Control	MGGC	52	53	0030h	RO;
Mirror of Dev0 DEVEN	MDEVENdev0F 0	54	57	[Device Specific]	RO;
Software Scratch Read Write	SSRW	58	5B	00000000h	R/W;
Base of Stolen Memory	BSM	5C	5F	[Device Specific]	RO;
Hardware Scratch Read Write	HSRW	60	61	0000h	R/W;
Multi Size Aperture Control	MSAC	62	62	02h	RO; R/W; R/W/L;
VTD Status	VTDS	63	63	02h or 00h	RO;
Secondary CWB Flush Control [DevBW Only]	SCWBFC	68	6F	000000000000 0000h	RO
Capabilities List Control	CAPL	7F	7F	00h	RO; R/W;
Message Signaled Interrupts Capability ID	MSI_CAPID	90	91	D005h	RO;
Message Control	MC	92	93	0000h	RO; R/W;
Message Address	MA	94	97	00000000h	R/W; RO;
Message Data	MD	98	99	0000h	R/W;
FLR Capability ID	FLRCAPID	A4	A5	0009h	RO;
FLR Length and Version	FLRLENVER	A6	A7	2006h	RO;
FLR Control	FLRCNTL	A8	A9	0000h	RO; R/W;
FLR Status	FLRSTAT	AA	AA	00h	RO
Graphics Device Reset	GDRST	C0	C0	00h	RO; R/W;
GMBUS frequency binary encoding	GMBUSFREQ	CC	CD	0000h	R/W; RO;
Power Management Capabilities ID	PMCAPID	D0	D1	0001h	RO;

Register Name	Register Symbol	Register Start	Register End	Default Value	Access
Power Management Capabilities	PMCAP	D2	D3	0022h or 0023h	RO;
Power Management Control/Status	PMCS	D4	D5	0000h	RO; R/W;
Software SMI	SWSMI	E0	E1	0000h	R/W;
System Display Event Register	ASLE	E4	E7	00000000h	R/W;
Software SCI	SWSCI	E8	E9	0000h	RO; R/W;
Legacy Backlight Brightness	LBB	F4	F7	00000000h	R/W;
Manufacturing ID	MID2	F8	FB	[Device Specific]	RO;
ASL Storage	ASLS	FC	FF	00000000h	R/W;

7.2.1 VID2 — Vendor Identification

B/D/F/Type:	0/2/0/PCI
Address Offset:	0-1h
Default Value:	8086h
Access:	RO;
Size:	16 bits

This register combined with the Device Identification register uniquely identifies any PCI device.

Bit	Access	Default Value	Description
15:0	RO	8086h	**Vendor Identification Number (VID):** PCI standard identification for Intel.

7.2.2 DID2 — Device Identification

B/D/F/Type:	0/2/0/PCI
Address Offset:	2-3h
Default Value:	[Device Specific]
Access:	RO;
Size:	16 bits

This register combined with the Vendor Identification register uniquely identifies any PCI device.

Bit	Access	Default Value	Description
15:0	RO	--	**Device Identification Number (DID):** Identifier assigned to the GMCH core/primary PCI device. Intel Reserved Text: Some bits of this field are actually determined by fuses, which allows unique Device IDs to be used for different product SKUs.

7.2.3 PCICMD2 — PCI Command

B/D/F/Type:	0/2/0/PCI
Address Offset:	4-5h
Default Value:	0000h
Access:	RO; R/W;
Size:	16 bits

This 16-bit register provides basic control over the IGDs ability to respond to PCI cycles. The PCICMD Register in the IGD disables the IGD PCI compliant master accesses to main memory.

Bit	Access	Default Value	Description
15:11	RO	00h	**Reserved**
10	R/W	0b	**Interrupt Disable:** This bit disables the device from asserting INTx#. 0: Enable the assertion of this device's INTx# signal. 1: Disable the assertion of this device's INTx# signal. DO_INTx messages will not be sent to DMI.
9	RO	0b	**Fast Back-to-Back (FB2B):** Not Implemented. Hardwired to 0.
8	RO	0b	**SERR Enable (SERRE):** Not Implemented. Hardwired to 0.
7	RO	0b	**Address/Data Stepping Enable (ADSTEP):** Not Implemented. Hardwired to 0.
6	RO	0b	**Parity Error Enable (PERRE):** Not Implemented. Hardwired to 0. Since the IGD belongs to the category of devices that does not corrupt programs or data in system memory or hard drives, the IGD ignores any parity error that it detects and continues with normal operation.
5	RO	0b	**Video Palette Snooping (VPS):** This bit is hardwired to 0 to disable snooping.
4	RO	0b	**Memory Write and Invalidate Enable (MWIE):** Hardwired to 0. The IGD does not support memory write and invalidate commands.
3	RO	0b	**Special Cycle Enable (SCE):** This bit is hardwired to 0. The IGD ignores Special cycles.
2	R/W	0b	**Bus Master Enable (BME):** 0: Disable IGD bus mastering. 1: Enable the IGD to function as a PCI compliant master.
1	R/W	0b	**Memory Access Enable (MAE):** This bit controls the IGDs response to memory space accesses. 0: Disable. 1: Enable.
0	R/W	0b	**I/O Access Enable (IOAE):** This bit controls the IGDs response to I/O space accesses. 0: Disable. 1: Enable.

7.2.4 PCISTS2 — PCI Status

B/D/F/Type:	0/2/0/PCI
Address Offset:	6-7h
Default Value:	0090h
Access:	RO;
Size:	16 bits

PCISTS is a 16-bit status register that reports the occurrence of a PCI compliant master abort and PCI compliant target abort. PCISTS also indicates the DEVSEL# timing that has been set by the IGD.

Bit	Access	Default Value	Description
15	RO	0b	**Detected Parity Error (DPE):** Since the IGD does not detect parity, this bit is always hardwired to 0.g
14	RO	0b	**Signaled System Error (SSE):** The IGD never asserts SERR#, therefore this bit is hardwired to 0.
13	RO	0b	**Received Master Abort Status (RMAS):** The IGD never gets a Master Abort, therefore this bit is hardwired to 0.
12	RO	0b	**Received Target Abort Status (RTAS):** The IGD never gets a Target Abort, therefore this bit is hardwired to 0.
11	RO	0b	**Signaled Target Abort Status (STAS):** Hardwired to 0. The IGD does not use target abort semantics.
10:9	RO	00b	**DEVSEL Timing (DEVT):** N/A. These bits are hardwired to "00".
8	RO	0b	**Master Data Parity Error Detected (DPD):** Since Parity Error Response is hardwired to disabled (and the IGD does not do any parity detection), this bit is hardwired to 0.
7	RO	1b	**Fast Back-to-Back (FB2B):** Hardwired to 1. The IGD accepts fast back-to-back when the transactions are not to the same agent.
6	RO	0b	**User Defined Format (UDF):** Hardwired to 0.
5	RO	0b	**66 MHz PCI Capable (66C):** N/A - Hardwired to 0.
4	RO	1b	**Capability List (CLIST):** This bit is set to 1 to indicate that the register at 34h provides an offset into the function痴 PCI Configuration Space containing a pointer to the location of the first item in the list.
3	RO	0b	**Interrupt Status:** This bit reflects the state of the interrupt in the device. Only when the Interrupt Disable bit in the command register is a 0 and this Interrupt Status bit is a 1, will the devices INTx# signal be asserted. Setting the Interrupt Disable bit to a 1 has no effect on the state of this bit.
2:0	RO	000b	**Reserved.:**

7.2.5 RID2 — Revision Identification

B/D/F/Type:	0/2/0/PCI
Address Offset:	8h
Default Value:	00h
Access:	RO;
Size:	8 bits

Compatible Revision ID (CRID):

An 8 bit hardwired value assigned by the ID Council. Normally, the value assigned as the CRID will be identical to the SRID value of a previous stepping of the product with which the new product is deemed "compatible". Note that CRID is not an addressable PCI register. The CRID value is simply reflected through the RID register when appropriately selected. Lower 4 bits of the CRID are driven by Fuses. The CRID fuses are programmed based on the SKU.

Stepping Revision ID (SRID):

An 8 bit hardwired value assigned by the ID Council. The values assigned as the SRID of a product's steppings will be selectively incremented based on the degree of change to that stepping. It is suggested that the first stepping of any given product have an SRID value = 01h simply to avoid the "reserved register" value of 00h. Note that SRID is not an addressable PCI register. The SRID value is simply reflected through the RID register when appropriately selected.

RID Select Key Value:

This is hardwired value (69h). If the latched value written to the RID register address matches this RID Select Key Value, the CRID value be presented for reading from the RID register.

RID Definition:

This register contains the revision number of the GMCH Device #0. Following PCI Reset the SRID value is selected to be read. When a write occurs to this register the write data is compared to the hardwired RID Select Key Value which is 69h. If the data matches this key a flag is set that enables the CRID value to be read through this register.

Note that the flag is a "write once'. Therefore once the CRID is selected to be read, the only way to again select the SRID is to PCI Reset the component. Also if any value other than the key (69h) is written to the RID register, the flag is locked such that the SRID is selected until the component is PCI Reset. Note that the RID register itself is not directly write-able.

This register contains the revision number for Device #2 Functions 0 and 1.

Bit	Access	Default Value	Description
7:0	RO	00h	**Revision Identification Number (RID):** This is an 8-bit value that indicates the revision identification number for the GMCH.

7.2.6 CC — Class Code

B/D/F/Type:	0/2/0/PCI
Address Offset:	9-Bh
Default Value:	030000h
Access:	RO;
Size:	24 bits

This register contains the device programming interface information related to the Sub-Class Code and Base Class Code definition for the IGD. This register also contains the Base Class Code and the function sub-class in relation to the Base Class Code.

Bit	Access	Default Value	Description
23:16	RO	03h	**Base Class Code (BCC):** This is an 8-bit value that indicates the base class code for the GMCH. This code has the value 03h, indicating a Display Controller.
15:8	RO	00h	**Sub-Class Code (SUBCC):** Based on Device #0 GGC-GMS bits and GGC-IVD bits. 00h: VGA compatible 80h: Non VGA (GMS = "000" or IVD = "1")
7:0	RO	00h	**Programming Interface (PI):** 00h: Hardwired as a Display controller.

7.2.7 CLS — Cache Line Size

B/D/F/Type:	0/2/0/PCI
Address Offset:	Ch
Default Value:	00h
Access:	RO;
Size:	8 bits

The IGD does not support this register as a PCI slave.

Bit	Access	Default Value	Description
7:0	RO	00h	**Cache Line Size (CLS):** This field is hardwired to 0s. The IGD as a PCI compliant master does not use the Memory Write and Invalidate command and, in general, does not perform operations based on cache line size.

7.2.8 MLT2 — Master Latency Timer

B/D/F/Type:	0/2/0/PCI
Address Offset:	Dh
Default Value:	00h
Access:	RO;
Size:	8 bits

The IGD does not support the programmability of the master latency timer because it does not perform bursts.

Bit	Access	Default Value	Description
7:0	RO	00h	**Master Latency Timer Count Value:** Hardwired to 0s.

7.2.9 HDR2 — Header Type

B/D/F/Type:	0/2/0/PCI
Address Offset:	Eh
Default Value:	80h
Access:	RO;
Size:	8 bits

This register contains the Header Type of the IGD.

Bit	Access	Default Value	Description
7	RO	1b	**Multi Function Status (MFunc):** Indicates if the device is a Multi-Function Device. The Value of this register is determined by Device #0, offset 54h, DEVEN[4]. If Device #0 DEVEN[4] is set, the Mfunc bit is also set.
6:0	RO	00h	**Header Code (H):** This is a 7-bit value that indicates the Header Code for the IGD. This code has the value 00h, indicating a type 0 configuration space format.

7.2.10 BIST — Built In Self Test

B/D/F/Type:	0/2/0/PCI
Address Offset:	Fh
Default Value:	00h
Access:	RO;
Size:	8 bits

This register is used for control and status of Built In Self Test (BIST).

Bit	Access	Default Value	Description
7	RO	0b	**BIST Supported:** BIST is not supported. This bit is hardwired to 0.
6:0	RO	00h	**Reserved**

7.2.11 GTTMMADR — Graphics Translation Table Range Address

B/D/F/Type:	0/2/0/PCI
Address Offset:	10-17h
Default Value:	0000000000000004h
Access:	RO; R/W;
Size:	64 bits

This register requests allocation for combined Graphics Translation Table and Memory Mapped Range. The allocation is split evenly between GTTADDR and MMIO, with MMIO coming first (lowest address) in the space.

For the Global GTT, GTTADDR is defined as part of a memory BAR in graphics device config space as an alias with which software writes values (PTEs) into the global Graphics Translation Table (GTT). Writing PTEs directly into the global GTT memory area is allowed.

Device	Total Allocation	GTTADDR Size	# GTT Entries	Total Aperture Size	Base Address Bits
All	**1 MB**	**512K**	**128K**	**512M**	**35:20**

The device snoops writes to GTTADDR space in order to invalidate any cached translations within the various TLB's implemented on-chip. There are some exceptions to this – see GTT-TLB in the Programming Interface chapter.

The Global GTT base address is programmed in the PGTB_CNTL register. The Global GTT resides in Main Memory

The Global GTT is required to be 4KB aligned, with each entry being DWord aligned.

Bit	Access	Default Value	Description
63:36	R/W	0000000h	**Must be set to 0 since addressing above 64GB is not supported.**
35:21	R/W	0000h	**Memory Base Address:** Set by the OS, these bits correspond to address signals [35:21].
20	R/W		**R/W, Memory Base Address[20].** 0 indicates at least 2MB address range.
19:4	RO	0000h	**Reserved:** Hardwired to 0's to indicate at least 1MB address range.
3	RO	0b	**Prefetchable Memory:** Hardwired to 0 to prevent prefetching.
2:1	RO	10b	**Memory Type ()** 00 : To indicate 32 bit base address 01: Reserved 10 : To indicate 64 bit base address 11: Reserved
0	RO	0b	**Memory/IO Space:** Hardwired to 0 to indicate memory space.

7.2.12 GMADR — Graphics Memory Range Address

B/D/F/Type:	0/2/0/PCI
Address Offset:	18-1Fh
Default Value:	000000000000000Ch
Access:	RO; R/W; R/W/L;
Size:	64 bits

IGD graphics memory base address is specified in this register.

Bit	Access	Default Value	Description
63:36	RO	0000000h	**Reserved**
35:29	R/W	00h	**Memory Base Address:** Set by the OS, these bits correspond to address signals [35:29].
28	R/W/L	0b	**512 MB Address Mask:** This bit is either part of the Memory Base Address (R/W) or part of the Address Mask (RO), depending on the value of MSAC[1:0]. See MSAC (Dev 2, Func 0, offset 62h) for details.
27	R/W/L	0b	**256 MB Address Mask:** This bit is either part of the Memory Base Address (R/W) or part of the Address Mask (RO), depending on the value of MSAC[1:0]. See MSAC (Dev 2, Func 0, offset 62h) for details.
26:4	RO	000000h	**Address Mask:** Hardwired to 0s to indicate at least 128MB address range.
3	RO	1b	**Prefetchable Memory:** Hardwired to 1 to enable prefetching.
2:1	RO	10b	**Memory Type ()** 00 : To indicate 32 bit base address 01: Reserved 10 : To indicate 64 bit base address 11: Reserved
0	RO	0b	**Memory/IO Space:** Hardwired to 0 to indicate memory space.

7.2.13 IOBAR — I/O Base Address

B/D/F/Type:	0/2/0/PCI
Address Offset:	20-23h
Default Value:	00000001h
Access:	RO; R/W;
Size:	32 bits

This register provides the Base offset of the I/O registers within Device #2. Bits 15:3 are programmable allowing the I/O Base to be located anywhere in 16bit I/O Address Space. Bits 2:1 are fixed and return zero, bit 0 is hardwired to a one indicating that 8 bytes of I/O space are decoded.

Access to the 8Bs of IO space is allowed in PM state D0 when IO Enable (PCICMD bit 0) set. Access is disallowed in PM states D1-D3 or if IO Enable is clear or if Device #2 is turned off or if internal graphics is disabled thru the fuse or fuse override mechanisms. Note that access to this IO BAR is independent of VGA functionality within Device #2. Also note that this mechanism in available only through function 0 of Device#2 and is not duplicated in function #1.

If accesses to this IO bar are allowed then the GMCH claims all 8, 16 or 32 bit IO cycles from the CPU that falls within the 8B claimed.

Bit	Access	Default Value	Description
31:16	RO	0000h	**Reserved** Read as 0's, these bits correspond to address signals [31:16].
15:3	R/W	0000h	**IO Base Address:** Set by the OS, these bits correspond to address signals [15:3].
2:1	RO	00b	**Memory Type:** Hardwired to 0s to indicate 32-bit address.
0	RO	1b	**Memory / IO Space:** Hardwired to 1 to indicate IO space.

7.2.14 SVID2 — Subsystem Vendor Identification

B/D/F/Type:	0/2/0/PCI
Address Offset:	2C-2Dh
Default Value:	0000h
Access:	R/WO;
Size:	16 bits

Bit	Access	Default Value	Description
15:0	R/WO	0000h	**Subsystem Vendor ID:** This value is used to identify the vendor of the subsystem. This register should be programmed by BIOS during boot-up. Once written, this register becomes Read-Only. This register can only be cleared by a Reset.

7.2.15 SID2 — Subsystem Identification

B/D/F/Type:	0/2/0/PCI
Address Offset:	2E-2Fh
Default Value:	0000h
Access:	R/WO;
Size:	16 bits

Bit	Access	Default Value	Description
15:0	R/WO	0000h	**Subsystem Identification:** This value is used to identify a particular subsystem. This field should be programmed by BIOS during boot-up. Once written, this register becomes Read-Only. This register can only be cleared by a Reset.

7.2.16 ROMADR — Video BIOS ROM Base Address

B/D/F/Type:	0/2/0/PCI
Address Offset:	30-33h
Default Value:	00000000h
Access:	RO;
Size:	32 bits

The IGD does not use a separate BIOS ROM, therefore this register is hardwired to 0s.

Bit	Access	Default Value	Description
31:18	RO	0000h	**ROM Base Address:** Hardwired to 0s.
17:11	RO	00h	**Address Mask:** Hardwired to 0s to indicate 256 KB address range.
10:1	RO	000h	**Reserved:** Hardwired to 0s.
0	RO	0b	**ROM BIOS Enable:** 0 = ROM not accessible.

7.2.17 CAPPOINT — Capabilities Pointer

B/D/F/Type:	0/2/0/PCI
Address Offset:	34h
Default Value:	90h
Access:	RO;
Size:	8 bits

Bit	Access	Default Value	Description
7:0	RO	90h	**Capabilities Pointer Value:** This field contains an offset into the function's PCI Configuration Space for the first item in the New Capabilities Linked List which is the MSI Capabilities ID register at address 90h or the Power Management Capabilities ID registers at address D0h. The value is determined by CAPL[0].

7.2.18 INTRLINE — Interrupt Line

B/D/F/Type:	0/2/0/PCI
Address Offset:	3Ch
Default Value:	00h
Access:	R/W;
Size:	8 bits

Bit	Access	Default Value	Description
7:0	R/W	00h	**Interrupt Connection:** Used to communicate interrupt line routing information. POST software writes the routing information into this register as it initializes and configures the system. The value in this register indicates which input of the system interrupt controller that the device's interrupt pin is connected to.

7.2.19 INTRPIN — Interrupt Pin

B/D/F/Type:	0/2/0/PCI
Address Offset:	3Dh
Default Value:	01h
Access:	RO;
Size:	8 bits

Bit	Access	Default Value	Description
7:0	RO	01h	**Interrupt Pin:** As a single function device, the IGD specifies INTA# as its interrupt pin. 01h: INTA#.

7.2.20 MINGNT — Minimum Grant

B/D/F/Type:	0/2/0/PCI
Address Offset:	3Eh
Default Value:	00h
Access:	RO;
Size:	8 bits

Bit	Access	Default Value	Description
7:0	RO	00h	**Minimum Grant Value:** The IGD does not burst as a PCI compliant master.

7.2.21 MAXLAT — Maximum Latency

B/D/F/Type:	0/2/0/PCI
Address Offset:	3Fh
Default Value:	00h
Access:	RO;
Size:	8 bits

Bit	Access	Default Value	Description
7:0	RO	00h	**Maximum Latency Value:** The IGD has no specific requirements for how often it needs to access the PCI bus.

7.2.22 MCAPPTR — Capabilities Pointer (to Mirror of Dev0 CAPID)

B/D/F/Type:	0/2/0/PCI
Address Offset:	44h
Default Value:	48h
Access:	RO;
Size:	8 bits

Bit	Access	Default Value	Description
7:0	RO	48h	**Capabilities Pointer Value:** In this case the first capability is the product-specific Capability Identifier (CAPID0).

7.2.23 MCAPID — Mirror of Dev 0 Capability Identification.

B/D/F/Type:	0/2/0/PCI
Address Offset:	48-51h
Default Value:	[Device Specific]
Access:	RO;
Size:	80 bits

This is an INTEL RESERVED register and should NOT be disclosed to customers. It is for test and debug purposes only and will not be included in external documentation. Control of bits in this register are only required for customer visible SKU differentiation.

Bit	Access	Default Value	Description
79:0	RO	--	Device Specific Bit Definitions

7.2.24 MGGC — Mirror of Dev0 GMCH Graphics Control

B/D/F/Type:	0/2/0/PCI
Address Offset:	52-53h
Default Value:	0030h
Access:	RO;
Size:	16 bits

All the Bits in this register are LT locked.

Bit	Access	Default Value	Description
15:7	RO	000000000b	**Reserved**
6:4	RO	011b	**Graphics Mode Select (GMS):** This field is used to select the amount of Main Memory that is pre-allocated to support the Internal Graphics device in VGA (non-linear) and Native (linear) modes. The BIOS ensures that memory is pre-allocated only when Internal graphics is enabled. Stolen Memory Base is located between (TOLUD - SMSize) to TOUD. 000 = No memory pre-allocated. Device 2 (IGD) does not claim VGA cycles (Mem and IO), and the Sub-Class Code field within Device 2 function 0. Class Code register is 80. 001 = DVMT (UMA) mode, 1 MB memory pre-allocated for frame buffer. 010 = DVMT (UMA) mode, 4 MB memory pre-allocated for frame buffer. 011 = DVMT (UMA) mode, 8 MB memory pre-allocated for frame buffer. 100 = DVMT (UMA) mode, 16 MB memory pre-allocated for frame buffer. 101 = DVMT (UMA) mode, 32 MB memory pre-allocated for frame buffer. 110 = DVMT (UMA) mode, 48 MB memory pre-allocated for frame buffer. 111 = DVMT (UMA) mode, 64 MB memory pre-allocated for frame buffer. Note: This register is locked and becomes Read Only when the D_LCK bit in the SMRAM register is set. Hardware does not clear or set any of these bits automatically based on IGD being disabled/enabled.
3:2	RO	00b	**Reserved**
1	RO	0b	**IGD VGA Disable (IVD):** 1:Disable. Device 2 (IGD) does not claim VGA cycles (Mem and IO), and the Sub-Class Code field within Device 2 function 0 Class Code register is 80. 0: Enable (Default). Device 2 (IGD) claims VGA memory and IO cycles, the Sub-Class Code within Device 2 Class Code register is 00.
0	RO	0b	**Reserved**

7.2.25 MDEVENdev0F0 — Mirror of Dev0 DEVEN

B/D/F/Type:	0/2/0/PCI
Address Offset:	54-57h
Default Value:	[Device Specific]
Access:	RO;
Size:	32 bits

Allows for enabling/disabling of PCI devices and functions that are within the MCH. All the Bits in this register are LT locked.

Bit	Access	Default Value	Description
31:0	RO	--	Device Specific Bit Definitions. See Device 0 documentation in the EDS.

7.2.26 SSRW — Software Scratch Read Write

B/D/F/Type:	0/2/0/PCI
Address Offset:	58-5Bh
Default Value:	00000000h
Access:	R/W;
Size:	32 bits

Bit	Access	Default Value	Description
31:0	R/W	00000000h	**Reserved**

7.2.27 BSM — Base of Stolen Memory

B/D/F/Type:	0/2/0/PCI
Address Offset:	5C-5Fh
Default Value:	[Device Specific]
Access:	RO;
Size:	32 bits

Graphics Stolen Memory and TSEG are within DRAM space defined under TOLUD. From the top of low used DRAM, GMCH claims 1 to 64MBs of DRAM for internal graphics if enabled.

Bit	Access	Default Value	Description
31:20	RO	--	**Base of Stolen Memory (BSM):** This register contains bits 31 to 20 of the base address of stolen DRAM memory. The host interface determines the base of graphics stolen memory by subtracting the graphics stolen memory size from TOLUD. See Device 0 TOLUD in the EDS for more explanation.
19:0	RO	00000h	**Reserved**

7.2.28 HSRW — Hardware Scratch Read Write

B/D/F/Type:	0/2/0/PCI
Address Offset:	60-61h
Default Value:	0000h
Access:	R/W;
Size:	16 bits

Bit	Access	Default Value	Description
15:0	R/W	0000h	**Reserved R/W**

7.2.29 MSAC — Multi Size Aperture Control

B/D/F/Type:	0/2/0/PCI
Address Offset:	62h
Default Value:	02h
Access:	RO; R/W; R/W/L;
Size:	8 bits

This register determines the size of the graphics memory aperture in function 0. By default the aperture size is 256 MB. Only the system BIOS will write this register based on pre-boot address allocation efforts, but the graphics may read this register to determine the correct aperture size. System BIOS needs to save this value on boot so that it can reset it correctly during S3 resume.

Bit	Access	Default Value	Description
7:4	R/W	0h	**Scratch Bits Only:** Have no physical effect on hardware.
3	RO	0b	**Reserved**
2:1	R/W/L	01b	**Aperture Size (LHSAS):** 11: bits [28:27] of GMADR register are made Read only and forced to zero, allowing only 512MB of GMADR 01: bit [28] of GMADR is made R/W and bit [27] of GMADR is forced to zero allowing 256MB of GMADR 00: bits [28:27] of GMADR register are made R/W allowing 128MB of GMADR 10: Illegal programming. ***This bit is LT locked, becomes read-only when trusted environment is launched.***
0	RO	0b	**Reserved**

7.2.30 SCWBFC — Secondary CWB Flush Control ([DevBW] Only)

B/D/F/Type:	0/2/0/PCI
Address Offset:	68-6Fh
Default Value:	0000000000000000h
Access:	W
Size:	64 bits

This register is for hardware debug purposes only. This is not relevant for software. All the data stored in the secondary CWB is flushed to memory before a write to this page is completed on the Front side bus. The write data is discarded. All transactions from the CPU that follow are not processed by the chipset till the "flush write" completes creating a fence beyond which coherency is guaranteed.

A read to this page does not flush the primary CWB/DWB and returns Zeros.

Bit	Access	Default Value	Description
63:0	W	0000000000000000h	**Secondary CWB Flush Control (SCWBFC):** A CPU Dword/Qword write to this space flushes the Secondary CWB/DWB of all writes. The data is discarded..

7.2.31 CAPL — Capabilities List Control

B/D/F/Type:	0/2/0/PCI
Address Offset:	7Fh
Default Value:	00h
Access:	RO; R/W;
Size:	8 bits

Allows BIOS to hide capabilities that are part of the Device 2 PCI Capabilities Linked List. By setting the appropriate bits, certain capabilities will be "skipped" during a later phase of system initialization. This is an INTEL RESERVED register and should NOT be disclosed to customers. It is for test and debug purposes only and will not be included in external documentation.

Bit	Access	Default Value	Description
7:1	RO	00h	**Reserved.:**
0	R/W	0b	**MSI Capability Hidden (MSICH):** 0: MSI Capability at 90h is included in list. 1: MSI Capability is NOT included in list. Power Management Capability ID's (D0h) pointer is the next capability.

7.2.32 MSI_CAPID — Message Signaled Interrupts Capability ID

B/D/F/Type:	0/2/0/PCI
Address Offset:	90-91h
Default Value:	D005h
Access:	RO;
Size:	16 bits

When a device supports MSI it can generate an interrupt request to the processor by writing a predefined data item (a message) to a predefined memory address. The reporting of the existence of this capability can be disabled by setting MSICH (CAPL[0] @ 7Fh). In that case walking this linked list will skip this capability and instead go directly to the PCI PM capability.

Bit	Access	Default Value	Description
15:8	RO	D0h	**Pointer to Next Capability:** This contains a pointer to the next item in the capabilities list which is the Power Management Capability.
7:0	RO	05h	**Capability ID:** Value of 05h identifies this linked list item (capability structure) as being for MSI registers.

7.2.33 MC — Message Control

B/D/F/Type:	0/2/0/PCI
Address Offset:	92-93h
Default Value:	0000h
Access:	RO; R/W;
Size:	16 bits

System software can modify bits in this register, but the device is prohibited from doing so.
If the device writes the same message multiple times, only one of those messages is guaranteed to be serviced. If all of them must be serviced, the device must not generate the same message again until the driver services the earlier one.

Bit	Access	Default Value	Description
15:8	RO	00h	**Reserved**
7	RO	0b	**64-bit Address Capable:** Hardwired to 0 to indicate that the function does not implement the upper 32 bits of the Message Address register and is incapable of generating a 64-bit memory address. This may need to change in future implementations when addressable system memory exceeds the 32bit/4GB limit.
6:4	R/W	000b	**Multiple Message Enable (MME):** System software programs this field to indicate the actual number of messages allocated to this device. This number will be equal to or less than the number actually requested. The encoding is the same as for the MMC field below.
3:1	RO	000b	**Multiple Message Capable (MMC):** System software reads this field to determine the number of messages being requested by this device. Value : Number of Messages Requested

Bit	Access	Default Value	Description
			000: 1 All of the following are reserved in this implementation: 001: 2 010: 4 011: 8 100: 16 101: 32 110: Reserved. 111: Reserved.
0	R/W	0b	**MSI Enable (MSIEN):** Controls the ability of this device to generate MSIs.

7.2.34 MA — Message Address

B/D/F/Type:	0/2/0/PCI
Address Offset:	94-97h
Default Value:	00000000h
Access:	R/W; RO;
Size:	32 bits

A read from this register produces undefined results.

Bit	Access	Default Value	Description
31:2	R/W	00000000h	**Message Address:** Used by system software to assign an MSI address to the device. The device handles an MSI by writing the padded contents of the MD register to this address.
1:0	RO	00b	**Force Dword Align:** Hardwired to 0 so that addresses assigned by system software are always aligned on a dword address boundary.

7.2.35 MD — Message Data

B/D/F/Type:	0/2/0/PCI
Address Offset:	98-99h
Default Value:	0000h
Access:	R/W;
Size:	16 bits

Bit	Access	Default Value	Description
15:0	R/W	0000h	**Message Data:** Base message data pattern assigned by system software and used to handle an MSI from the device. When the device must generate an interrupt request, it writes a 32-bit value to the memory address specified in the MA register. The upper 16 bits are always set to 0. The lower 16 bits are supplied by this register.

7.2.36 GDRST — Graphics Device Reset

B/D/F/Type:	0/2/0/PCI
Address Offset:	C0h
Default Value:	00h
Access:	RO; RW/L;
Size:	8 bits

Bit	Access	Default Value	Description
7:5	RO	0h	**Reserved ():**
4:2	RW/L	00b	**Graphics Reset Domain (GRDOM):** Graphics Reset Domain (GRDOM): 000 – Full Graphics Reset will be performed (Render and Media engines and Display clock domain resets asserted) 001 – Render Engine only will be reset 011 – Media Engine only will be reset 010 – Reserved (Illegal Programming) 1XX – Reserved (Illegal Programming)
1	RO	0h	**Reserved ():**
0	RW/L	0b	**Graphics Reset Enable (GR):** Setting this bit asserts graphics-only reset. The clock domains to be reset are determined by GRDOM. Hardware resets this bit when the reset is complete. Setting this bit without waiting for it to clear, is undefined behavior. Once this bit is set to a "1" all MMIO registers associated with the selected engine(s) are returned to power on default state. Ring buffer pointers are reset, command stream fetches are dropped and ongoing render pipeline processing is halted, state machines and State Variables returned to power on default state. If the Display is reset, all display engines are halted (garbage on screen). VGA memory is not available, Store DWORDs and interrupts associated with the reset engine(s) are not guaranteed to be completed. Device #2 IO registers are not available. Device #2 Config registers continue to be available while Graphics reset is asserted.

7.2.37 GMBUSFREQ — GMBUS frequency binary encoding

B/D/F/Type:	0/2/0/PCI
Address Offset:	CC-CDh
Default Value:	0000h
Access:	R/W; RO;
Size:	16 bits

Bit	Access	Default Value	Description
15:10	RO	000000b	**Reserved (RSVD)**
9:0	R/W	0000000 000b	**CMBUS CDCLK frequency (cdfreq)**

7.2.38 PMCAPID — Power Management Capabilities ID

B/D/F/Type:	0/2/0/PCI
Address Offset:	D0-D1h
Default Value:	0001h
Access:	RO;
Size:	16 bits

Bit	Access	Default Value	Description
15:8	RO	00h	**NEXT_PTR:** This contains a pointer to the next item in the capabilities list.
7:0	RO	01h	**CAP_ID:** SIG defines this ID is 01h for power management.

7.2.39 PMCAP — Power Management Capabilities

B/D/F/Type:	0/2/0/PCI
Address Offset:	D2-D3h
Default Value:	0022h
Access:	RO;
Size:	16 bits

Bit	Access	Default Value	Description
15:11	RO	00h	**PME Support:** This field indicates the power states in which the IGD may assert PME#. Hardwired to 0 to indicate that the IGD does not assert the PME# signal.
10	RO	0b	**D2:** The D2 power management state is not supported. This bit is hardwired to 0.
9	RO	0b	**D1:** Hardwired to 0 to indicate that the D1 power management state is not supported.
8:6	RO	000b	**Reserved.**
5	RO	1b	**Device Specific Initialization (DSI):** Hardwired to 1 to indicate that special initialization of the IGD is required before generic class device driver is to use it.
4	RO	0b	**Auxiliary Power Source:** Hardwired to 0.
3	RO	0b	**PME Clock:** Hardwired to 0 to indicate IGD does not support PME# generation.
2:0	RO	01-b	**Version:** [DevBW] Hardwired to 010b to indicate that there are 4 bytes of power management registers implemented and that this device complies with revision 1.1 of the PCI Power Management Interface Specification. [DevCL] 010b indicates compliant with revision 1.1 of the PCI Power Management Speficiation. 011b indicates compliance with revision 1.2 of the PCI Power Management Specification. The value in this register is determined by the value of MCHBAR offset C08[15].

7.2.40 PMCS — Power Management Control/Status

B/D/F/Type:	0/2/0/PCI
Address Offset:	D4-D5h
Default Value:	0000h
Access:	RO; R/W;
Size:	16 bits

Bit	Access	Default Value	Description
15	RO	0b	**PME_Status:** This bit is 0 to indicate that IGD does not support PME# generation from D3 (cold).
14:13	RO	00b	**Data Scale (Reserved):** The IGD does not support data register. This bit always returns 0 when read, write operations have no effect.
12:9	RO	0h	**Data_Select (Reserved):** The IGD does not support data register. This bit always returns 0 when read, write operations have no effect.
8	RO	0b	**PME_En:** This bit is 0 to indicate that PME# assertion from D3 (cold) is disabled.
7:4	RO	00h	**Reserved** Always returns 0 when read, write operations have no effect.
3	RO	-	[DevBW] Only: Reserved, hardwired to 0. **No_Soft_Reset.** Will be set according to the state of MCHBAR C08[14]. When transitioning from D3hot to D0, a 0 indicates the device performs an internal reset, a 1 indicates that the device does not perform an internal reset, and Configuration Context is preserved. Note that the state of this bit has no hardware effect – it is programmable since there is some ambiguity as to which definition of the bit our hardware behavior better matches.
2	RO	0b	**Reserved** Always returns 0 when read, write operations have no effect.
1:0	R/W	00b	**PowerState:** This field indicates the current power state of the IGD and can be used to set the IGD into a new power state. If software attempts to write an unsupported state to this field, write operation must complete normally on the bus, but the data is discarded and no state change occurs. On a transition from D3 to D0 the graphics controller is optionally reset to initial values. Behavior of the graphics controller in supported states is detailed in the power management section of the PRM. Bits[1:0] Power state 00 D0 Default 01 D1 Not Supported 10 D2 Not Supported 11 D3

7.2.41 SWSMI — Software SMI

B/D/F/Type:	0/2/0/PCI
Address Offset:	E0-E1h
Default Value:	0000h
Access:	R/W; R/WC;
Size:	16 bits

Bit	Access	Default Value	Description
15:8	R/W	00h	**SW scratch bits:**
7:1	R/W	00h	**Software Flag:** Used to indicate caller and SMI function desired, as well as return result.
0	R/W	0b	**GMCH Software SMI Event:** When set this bit will trigger an SMI. Software must clear this bit to remove the SMI condition.

7.2.42 ASLE — System Display Event Register

B/D/F/Type:	0/2/0/PCI
Address Offset:	E4-E7h
Default Value:	00000000h
Access:	R/W;
Size:	32 bits

The exact use of these bytes including whether they are addressed as bytes,words, or as a dword, is not pre-determined but subject to change by driver and System BIOS teams (acting in unison).

Bit	Access	Default Value	Description
31:24	R/W	00h	**ASLE Scratch Trigger3:** When written, this scratch byte triggers an interrupt when IEF bit 0 is enabled and IMR bit 0 is unmasked. If written as part of a 16-bit or 32-bit write, only one interrupt is generated in common.
23:16	R/W	00h	**ASLE Scratch Trigger2:** When written, this scratch byte triggers an interrupt when IEF bit 0 is enabled and IMR bit 0 is unmasked. If written as part of a 16-bit or 32-bit write, only one interrupt is generated in common.
15:8	R/W	00h	**ASLE Scratch Trigger 1:** When written, this scratch byte triggers an interrupt when IEF bit 0 is enabled and IMR bit 0 is unmasked. If written as part of a 16-bit or 32-bit write, only one interrupt is generated in common.
7:0	R/W	00h	**ASLE Scratch Trigger 0:** When written, this scratch byte triggers an interrupt when IEF bit 0 is enabled and IMR bit 0 is unmasked. If written as part of a 16-bit or 32-bit write, only one interrupt is generated in common.

7.2.43 SWSCI — Software SCI

B/D/F/Type: 0/2/0/PCI
Address Offset: E8-E9h
Default Value: 0000h
Access: RWO; RW;
Size: 16 bits

This register serves 2 purposes:

1) Support selection of SMI or SCI event source (SMISCISEL - bit15)

2) SCI Event trigger (GSSCIE – bit 0).

To generate a SW SCI event, software (System BIOS/Graphics driver) should program bit 15 (SMISCISEL) to 1. This is typically programmed once (assuming SMIs are never triggered). On a "0" to "1" subsequent transition in bit 0 of this register (caused by a software write operation), GMCH sends a single SCI message (as currently defined in Grantsdale GMCH EDS) down the DMI link to ICH. ICH will set the DMISCI bit in its TCO1_STS register and TCOSCI_STS bit in its GPE0 register upon receiving this message from DMI. The corresponding SCI event handler in BIOS is to be defined as a _Lxx method, indicating level trigger to the operating system.

Once written as 1, software must write a "0" to this bit to clear it, and all other write transitions (1->0, 0->0, 1->1) or if bit 15 is "0" will not cause GMCH to send SCI message to DMI link.

To generate a SW SMI event, software should program bit 15 to 0 and trigger SMI via writes to SWSMI register (See SWSMI register for programming details).

Bit	Access	Default Value	Description
15	RWO	0b	**SMI or SCI event select (SMISCISEL):** SMI or SCI event select (SMISCISEL)- 0 = SMI (default) 1 = SCI If selected event source is SMI, SMI trigger and associated scratch bits accesses are performed via SWSMI register at offset E0h. If SCI event source is selected, the rest of the bits in this register provide SCI trigger capability and associated SW scratch pad area.
14:1	RW	00000000 000000b	**Software scratch bits (SCISB):** SW scratch bits (read/write bits not used by hardware) (SCISB)
0	RW	0b	**GMCH Software SCI Event (GSSCIE):** If SCI event is selected (SMISCISEL = 1), on a “0” to “1” transition of GSSCIE bit, GMCH will send a SCI message via DMI link to ICH to cause the TCOSCI_STS bit in its GPE0 register to be set to 1. Software must write a “0” to clear this bit.

7.2.44 LBB — Legacy Backlight Brightness ([DevCL] Only)

B/D/F/Type:	0/2/0/PCI
Address Offset:	F4-F7h
Default Value:	00000000h
Access:	R/W;
Size:	32 bits

This register can be accessed by either Byte, Word, or Dword PCI config cycles. A write to this register will cause the Backlight Event (Display B Interrupt) if enabled.

Bit	Access	Default Value	Description
31:24	R/W	00h	**LBPC Scratch Trigger3:** When written, this scratch byte triggers an interrupt when LBEE is enabled in the Pipe B Status register and the Display B Event is enabled in IER and unmasked in IMR etc. If written as part of a 16-bit or 32-bit write, only one interrupt is generated in common.
23:16	R/W	00h	**LBPC Scratch Trigger2:** When written, this scratch byte triggers an interrupt when LBEE is enabled in the Pipe B Status register and the Display B Event is enabled in IER and unmasked in IMR etc. If written as part of a 16-bit or 32-bit write, only one interrupt is generated in common.
15:8	R/W	00h	**LBPC Scratch Trigger1:** When written, this scratch byte triggers an interrupt when LBEE is enabled in the Pipe B Status register and the Display B Event is enabled in IER and unmasked in IMR etc. If written as part of a 16-bit or 32-bit write, only one interrupt is generated in common.
7:0	R/W	00h	**Legacy Backlight Brightness (LBES):** The value of zero is the lowest brightness setting and 255 is the brightest. A write to this register will cause a flag to be set (LBES) in the PIPEBSTATUS register and cause an interrupt if Backlight event in the PIPEBSTATUS register and cause an Interrupt if Backlight Event (LBEE) and Display B Event is enabled by software.

7.2.45 MID2 — Manufacturing ID

B/D/F/Type:	0/2/0/PCI
Address Offset:	F8-FBh
Default Value:	[Device Specific]
Access:	RO;
Size:	32 bits

This is an INTEL RESERVED register and should NOT be disclosed to customers. It is for test and debug purposes only and will not be included in external documentation.

Bit	Access	Default Value	Description
31:24	RO	00h	**Reserved**
23:16	RO	--	**Manufacturing Stepping ID (MSTEP)**
15:8	RO	0Fh	**Foundry Code (FOUND):** 0Fh: Foundry code for Intel others: Reserved These bits identify the Foundry; code of 0000 1111b = foundry code for Intel.
7:3	RO	--	**Process ID (PROC)**
2:0	RO	--	**Dot Process (DOT)**

7.2.46 ASLS — ASL Storage

B/D/F/Type:	0/2/0/PCI
Address Offset:	FC-FFh
Default Value:	00000000h
Access:	R/W;
Size:	32 bits

This SW scratch register only needs to be read/write accessible. The exact bit register usage must be worked out in common between System BIOS and driver software, but storage for switching/indicating up to 6 devices is possible with this amount. For each device, the ASL control method will require two bits for _DOD (BIOS detectable yes or no, VGA/NonVGA), one bit for _DGS (enable/disable requested), and two bits for _DCS (enabled now/disabled now, connected or not).

Bit	Access	Default Value	Description
31:0	R/W	00000000h	RW according to a software controlled usage to support device switching.

7.3 Device 2, Function 1

Register Name	Register Symbol	Register Start	Register End	Default Value	Access
Vendor Identification	VID2	0	1	8086h	RO;
Device Identification	DID2	2	3	[Device Specific]	RO;
PCI Command	PCICMD2	4	5	0000h	RO; R/W;
PCI Status	PCISTS2	6	7	0090h	RO;
Revision Identification	RID2	8	8	00h	RO;
Class Code	CC	9	B	038000h	RO;
Cache Line Size	CLS	C	C	00h	RO;
Master Latency Timer	MLT2	D	D	00h	RO;
Header Type	HDR2	E	E	80h	RO;
Built In Self Test	BIST	F	F	00h	RO;
Memory Mapped Range Address	MMADR	10	17	0000000000000004h	RO; R/W;
Subsystem Vendor Identification	SVID2	2C	2D	0000h	RO;
Subsystem Identification	SID2	2E	2F	0000h	RO;
Video BIOS ROM Base Address	ROMADR	30	33	00000000h	RO;
Capabilities Pointer	CAPPOINT	34	34	D0h	RO;
Minimum Grant	MINGNT	3E	3E	00h	RO;
Maximum Latency	MAXLAT	3F	3F	00h	RO;
Capabilities Pointer (to Mirror of Dev0 CAPID)	MCAPPTR	44	44	48h	RO;
Mirror of Dev 0 Capability Identification	MCAPID	48	51	[Device Specific]	RO;
Mirror of Dev0 GMCH Graphics Control	MGGC	52	53	0030h	RO;
Mirror of Dev0 DEVEN	MDEVENdev0F0	54	57	[Device Specific]	RO;
Software Scratch Read Write	SSRW	58	5B	00000000h	RO;
Base of Stolen Memory	BSM	5C	5F	[Device Specific]	RO;
Hardware Scratch Read Write	HSRW	60	61	0000h	RO;
Multi Size Aperture Control	MSAC	62	62	02h	RO;

7.3.1 VID2 — Vendor Identification

B/D/F/Type:	0/2/0/PCI
Address Offset:	0-1h
Default Value:	8086h
Access:	RO;
Size:	16 bits

This register combined with the Device Identification register uniquely identifies any PCI device.

Bit	Access	Default Value	Description
15:0	RO	8086h	**Vendor Identification Number (VID):** PCI standard identification for Intel.

7.3.2 DID2 — Device Identification

B/D/F/Type:	0/2/0/PCI
Address Offset:	2-3h
Default Value:	[Device Specific]
Access:	RO;
Size:	16 bits

This register combined with the Vendor Identification register uniquely identifies any PCI device.

Bit	Access	Default Value	Description
15:0	RO	--	**Device Identification Number (DID):** Identifier assigned to the GMCH core/primary PCI device. Intel Reserved Text: Some bits of this field are actually determined by fuses, which allows unique Device IDs to be used for different product SKUs. See the device EDS for details.

7.3.3 PCICMD2 — PCI Command

B/D/F/Type:	0/2/0/PCI
Address Offset:	4-5h
Default Value:	0000h
Access:	RO; R/W;
Size:	16 bits

This 16-bit register provides basic control over the IGDs ability to respond to PCI cycles. The PCICMD Register in the IGD disables the IGD PCI compliant master accesses to main memory.

Bit	Access	Default Value	Description
15:11	RO	00h	**Reserved**
10	R/W	0b	**Interrupt Disable:** This bit disables the device from asserting INTx#. 0: Enable the assertion of this device's INTx# signal. 1: Disable the assertion of this device's INTx# signal. DO_INTx messages will not be sent to DMI.
9	RO	0b	**Fast Back-to-Back (FB2B):** Not Implemented. Hardwired to 0.
8	RO	0b	**SERR Enable (SERRE):** Not Implemented. Hardwired to 0.
7	RO	0b	**Address/Data Stepping Enable (ADSTEP):** Not Implemented. Hardwired to 0.
6	RO	0b	**Parity Error Enable (PERRE):** Not Implemented. Hardwired to 0. Since the IGD belongs to the category of devices that does not corrupt programs or data in system memory or hard drives, the IGD ignores any parity error that it detects and continues with normal operation.
5	RO	0b	**Video Palette Snooping (VPS):** This bit is hardwired to 0 to disable snooping.
4	RO	0b	**Memory Write and Invalidate Enable (MWIE):** Hardwired to 0. The IGD does not support memory write and invalidate commands.
3	RO	0b	**Special Cycle Enable (SCE):** This bit is hardwired to 0. The IGD ignores Special cycles.
2	R/W	0b	**Bus Master Enable (BME):** 0: Disable IGD bus mastering. 1: Enable the IGD to function as a PCI compliant master.
1	R/W	0b	**Memory Access Enable (MAE):** This bit controls the IGDs response to memory space accesses. 0: Disable. 1: Enable.
0	R/W	0b	**I/O Access Enable (IOAE):** This bit controls the IGDs response to I/O space accesses. 0: Disable. 1: Enable.

7.3.4 PCISTS2 — PCI Status

B/D/F/Type: 0/2/0/PCI
Address Offset: 6-7h
Default Value: 0090h
Access: RO;
Size: 16 bits

PCISTS is a 16-bit status register that reports the occurrence of a PCI compliant master abort and PCI compliant target abort. PCISTS also indicates the DEVSEL# timing that has been set by the IGD.

Bit	Access	Default Value	Description
15	RO	0b	**Detected Parity Error (DPE):** Since the IGD does not detect parity, this bit is always hardwired to 0.
14	RO	0b	**Signaled System Error (SSE):** The IGD never asserts SERR#, therefore this bit is hardwired to 0.
13	RO	0b	**Received Master Abort Status (RMAS):** The IGD never gets a Master Abort, therefore this bit is hardwired to 0.
12	RO	0b	**Received Target Abort Status (RTAS):** The IGD never gets a Target Abort, therefore this bit is hardwired to 0.
11	RO	0b	**Signaled Target Abort Status (STAS):** Hardwired to 0. The IGD does not use target abort semantics.
10:9	RO	00b	**DEVSEL Timing (DEVT):** N/A. These bits are hardwired to "00".
8	RO	0b	**Master Data Parity Error Detected (DPD):** Since Parity Error Response is hardwired to disabled (and the IGD does not do any parity detection), this bit is hardwired to 0.
7	RO	1b	**Fast Back-to-Back (FB2B):** Hardwired to 1. The IGD accepts fast back-to-back when the transactions are not to the same agent.
6	RO	0b	**User Defined Format (UDF):** Hardwired to 0.
5	RO	0b	**66 MHz PCI Capable (66C):** N/A - Hardwired to 0.
4	RO	1b	**Capability List (CLIST):** This bit is set to 1 to indicate that the register at 34h provides an offset into the function痴 PCI Configuration Space containing a pointer to the location of the first item in the list.
3	RO	0b	**Interrupt Status:** This bit reflects the state of the interrupt in the device. Only when the Interrupt Disable bit in the command register is a 0 and this Interrupt Status bit is a 1, will the devices INTx# signal be asserted. Setting the Interrupt Disable bit to a 1 has no effect on the state of this bit.
2:0	RO	000b	**Reserved.**

7.3.5 RID2 — Revision Identification

B/D/F/Type:	0/2/0/PCI
Address Offset:	8h
Default Value:	00h
Access:	RO;
Size:	8 bits

Compatible Revision ID (CRID):

An 8 bit hardwired value assigned by the ID Council. Normally, the value assigned as the CRID will be identical to the SRID value of a previous stepping of the product with which the new product is deemed "compatible". Note that CRID is not an addressable PCI register. The CRID value is simply reflected through the RID register when appropriately selected. Lower 4 bits of the CRID are driven by Fuses. The CRID fuses are programmed based on the SKU.

Stepping Revision ID (SRID):

An 8 bit hardwired value assigned by the ID Council. The values assigned as the SRID of a product's steppings will be selectively incremented based on the degree of change to that stepping. It is suggested that the first stepping of any given product have an SRID value = 01h simply to avoid the "reserved register" value of 00h. Note that SRID is not an addressable PCI register. The SRID value is simply reflected through the RID register when appropriately selected.

RID Select Key Value:

This is hardwired value (69h). If the latched value written to the RID register address matches this RID Select Key Value, the CRID value be presented for reading from the RID register.

RID Definition:

This register contains the revision number of the GMCH Device #0. Following PCI Reset the SRID value is selected to be read. When a write occurs to this register the write data is compared to the hardwired RID Select Key Value which is 69h. If the data matches this key a flag is set that enables the CRID value to be read through this register.

Note that the flag is a "write once'. Therefore once the CRID is selected to be read, the only way to again select the SRID is to PCI Reset the component. Also if any value other than the key (69h) is written to the RID register, the flag is locked such that the SRID is selected until the component is PCI Reset. Note that the RID register itself is not directly write-able.

This register contains the revision number for Device #2 Functions 0 and 1.

Bit	Access	Default Value	Description
7:0	RO	00h	**Revision Identification Number (RID):** This is an 8-bit value that indicates the revision identification number for the GMCH.

7.3.6 CC — Class Code

B/D/F/Type:	0/2/0/PCI
Address Offset:	9-Bh
Default Value:	038000h
Access:	RO;
Size:	24 bits

This register contains the device programming interface information related to the Sub-Class Code and Base Class Code definition for the IGD. This register also contains the Base Class Code and the function sub-class in relation to the Base Class Code.

Bit	Access	Default Value	Description
23:16	RO	03h	**Base Class Code (BCC):** This is an 8-bit value that indicates the base class code for the GMCH. This code has the value 03h, indicating a Display Controller.
15:8	RO	80h	**Sub-Class Code (SUBCC):** 80h: Non VGA
7:0	RO	00h	**Programming Interface (PI):** 00h: Hardwired as a Display controller.

7.3.7 CLS — Cache Line Size

B/D/F/Type:	0/2/0/PCI
Address Offset:	Ch
Default Value:	00h
Access:	RO;
Size:	8 bits

The IGD does not support this register as a PCI slave.

Bit	Access	Default Value	Description
7:0	RO	00h	**Cache Line Size (CLS):** This field is hardwired to 0s. The IGD as a PCI compliant master does not use the Memory Write and Invalidate command and, in general, does not perform operations based on cache line size.

7.3.8 MLT2 — Master Latency Timer

B/D/F/Type:	0/2/0/PCI
Address Offset:	Dh
Default Value:	00h
Access:	RO;
Size:	8 bits

The IGD does not support the programmability of the master latency timer because it does not perform bursts.

Bit	Access	Default Value	Description
7:0	RO	00h	**Master Latency Timer Count Value:** Hardwired to 0s.

7.3.9 HDR2 — Header Type

B/D/F/Type:	0/2/0/PCI
Address Offset:	Eh
Default Value:	80h
Access:	RO;
Size:	8 bits

This register contains the Header Type of the IGD.

Bit	Access	Default Value	Description
7	RO	1b	**Multi Function Status (MFunc):** Indicates if the device is a Multi-Function Device. The Value of this register is determined by Device #0, offset 54h, DEVEN[4]. If Device #0 DEVEN[4] is set, the Mfunc bit is also set.
6:0	RO	00h	**Header Code (H):** This is a 7-bit value that indicates the Header Code for the IGD. This code has the value 00h, indicating a type 0 configuration space format.

7.3.10 BIST — Built In Self Test

B/D/F/Type:	0/2/0/PCI
Address Offset:	Fh
Default Value:	00h
Access:	RO;
Size:	8 bits

This register is used for control and status of Built In Self Test (BIST).

Bit	Access	Default Value	Description
7	RO	0b	**BIST Supported:** BIST is not supported. This bit is hardwired to 0.
6:0	RO	00h	**Reserved**

7.3.11 MMADR — Memory Mapped Range Address

B/D/F/Type:	0/2/1/PCI
Address Offset:	10-17h
Default Value:	0000000000000004h
Access:	RO; R/W;
Size:	64 bits

This register requests allocation for the IGD registers and instruction ports. The allocation is for 512 KB and the base address is defined by bits [35:20].

Bit	Access	Default Value	Description
63:36	RO	0000000h	**Reserved ():**
35:20	R/W	0000h	**Memory Base Address ():** Set by the OS, these bits correspond to address signals [35:20].
19:4	RO	0000h	**Address Mask ():** Hardwired to 0's to indicate 512 KB address range (aligned to 1M boundary).
3	RO	0b	**Prefetchable Memory ():** Hardwired to 0 to prevent prefetching.
2	RO	1b	**Memory Type ():** 0 : To indicate 32 bit base address 1 : To indicate 64 bit base address
1	RO	0b	**Reserved ():**
0	RO	0b	**Memory / IO Space ():** Hardwired to 0 to indicate memory space.

7.3.12 SVID2 — Subsystem Vendor Identification

B/D/F/Type:	0/2/0/PCI
Address Offset:	2C-2Dh
Default Value:	0000h
Access:	RO;
Size:	16 bits

Bit	Access	Default Value	Description
15:0	RO	0000h	**Subsystem Vendor ID:** This value is used to identify the vendor of the subsystem. This register should be programmed by BIOS during boot-up. Once written, this register becomes Read-Only. This register can only be cleared by a Reset. NOTE: This is a RO copy of the Dev2Fxn0 value.

7.3.13 SID2 — Subsystem Identification

B/D/F/Type:	0/2/0/PCI
Address Offset:	2E-2Fh
Default Value:	0000h
Access:	RO;
Size:	16 bits

Bit	Access	Default Value	Description
15:0	RO	0000h	**Subsystem Identification:** This value is used to identify a particular subsystem. This field should be programmed by BIOS during boot-up. Once written, this register becomes Read-Only. This register can only be cleared by a Reset. NOTE: This is a RO copy of the Dev2Fxn0 value.

7.3.14 ROMADR — Video BIOS ROM Base Address

B/D/F/Type:	0/2/0/PCI
Address Offset:	30-33h
Default Value:	00000000h
Access:	RO;
Size:	32 bits

The IGD does not use a separate BIOS ROM, therefore this register is hardwired to 0s.

Bit	Access	Default Value	Description
31:18	RO	0000h	**ROM Base Address:** Hardwired to 0s.
17:11	RO	00h	**Address Mask:** Hardwired to 0s to indicate 256 KB address range.
10:1	RO	000h	**Reserved:** Hardwired to 0s.
0	RO	0b	**ROM BIOS Enable:** 0 = ROM not accessible.

7.3.15 CAPPOINT — Capabilities Pointer

B/D/F/Type:	0/2/0/PCI
Address Offset:	34h
Default Value:	D0h
Access:	RO;
Size:	8 bits

Bit	Access	Default Value	Description
7:0	RO	D0h	**Capabilities Pointer Value:** This field contains an offset into the function's PCI Configuration Space for the first item in the New Capabilities Linked List which the Power Management Capabilities ID registers at address D0h.

7.3.16 MINGNT — Minimum Grant

B/D/F/Type:	0/2/0/PCI
Address Offset:	3Eh
Default Value:	00h
Access:	RO;
Size:	8 bits

Bit	Access	Default Value	Description
7:0	RO	00h	**Minimum Grant Value:** The IGD does not burst as a PCI compliant master.

7.3.17 MAXLAT — Maximum Latency

B/D/F/Type:	0/2/0/PCI
Address Offset:	3Fh
Default Value:	00h
Access:	RO;
Size:	8 bits

Bit	Access	Default Value	Description
7:0	RO	00h	**Maximum Latency Value:** The IGD has no specific requirements for how often it needs to access the PCI bus.

7.3.18 MCAPPTR — Capabilities Pointer (to Mirror of Dev0 CAPID)

B/D/F/Type:	0/2/0/PCI
Address Offset:	44h
Default Value:	48h
Access:	RO;
Size:	8 bits

Bit	Access	Default Value	Description
7:0	RO	48h	**Capabilities Pointer Value:** In this case the first capability is the product-specific Capability Identifier (CAPID0).

7.3.19 MCAPID — Mirror of Dev 0 Capability Identification.

B/D/F/Type: 0/2/0/PCI
Address Offset: 48-51h
Default Value: [Device Specific]
Access: RO;
Size: 80 bits

This is an INTEL RESERVED register and should NOT be disclosed to customers. It is for test and debug purposes only and will not be included in external documentation. Control of bits in this register are only required for customer visible SKU differentiation.

Bit	Access	Default Value	Description
79:0	RO	--	Device Specific Bit Definitions – see the device EDS for details.

7.3.20 MGGC — Mirror of Dev0 GMCH Graphics Control

B/D/F/Type: 0/2/0/PCI
Address Offset: 52-53h
Default Value: 0030h
Access: RO;
Size: 16 bits

All the Bits in this register are LT locked.

Bit	Access	Default Value	Description
15:7	RO	000000000b	**Reserved**

Bit	Access	Default Value	Description
6:4	RO	011b	**Graphics Mode Select (GMS):** This field is used to select the amount of Main Memory that is pre-allocated to support the Internal Graphics device in VGA (non-linear) and Native (linear) modes. The BIOS ensures that memory is pre-allocated only when Internal graphics is enabled. Stolen Memory Base is located between (TOLUD - SMSize) to TOUD. 000 = No memory pre-allocated. Device 2 (IGD) does not claim VGA cycles (Mem and IO), and the Sub-Class Code field within Device 2 function 0. Class Code register is 80. 001 = DVMT (UMA) mode, 1 MB memory pre-allocated for frame buffer. 010 = DVMT (UMA) mode, 4 MB memory pre-allocated for frame buffer. 011 = DVMT (UMA) mode, 8 MB memory pre-allocated for frame buffer. 100 = DVMT (UMA) mode, 16 MB memory pre-allocated for frame buffer. 101 = DVMT (UMA) mode, 32 MB memory pre-allocated for frame buffer. 110 = DVMT (UMA) mode, 48 MB memory pre-allocated for frame buffer. 111 = DVMT (UMA) mode, 64 MB memory pre-allocated for frame buffer. Note: This register is locked and becomes Read Only when the D_LCK bit in the SMRAM register is set. Hardware does not clear or set any of these bits automatically based on IGD being disabled/enabled.
3:2	RO	00b	**Reserved**
1	RO	0b	**IGD VGA Disable (IVD):** 1 = Disable. Device 2 (IGD) does not claim VGA cycles (Mem and IO), and the Sub-Class Code field within Device 2 function 0 Class Code register is 80. 0 = Enable (Default). Device 2 (IGD) claims VGA memory and IO cycles, the Sub-Class Code within Device 2 Class Code register is 00.
0	RO	0b	**Reserved**

7.3.21 MDEVENdev0F0 — Mirror of Dev0 DEVEN

B/D/F/Type:	0/2/0/PCI
Address Offset:	54-57h
Default Value:	[Device Specific]
Access:	RO;
Size:	32 bits

Allows for enabling/disabling of PCI devices and functions that are within the MCH. All the Bits in this register are LT locked.

Bit	Access	Default Value	Description
31:0	RO	--	Device Specific Bit Definitions. See Device 0 documentation in the EDS.

7.3.22 SSRW — Software Scratch Read Write

B/D/F/Type:	0/2/0/PCI
Address Offset:	58-5Bh
Default Value:	00000000h
Access:	RO;
Size:	32 bits

Bit	Access	Default Value	Description
31:0	RO	00000000h	**Reserved**

7.3.22.1 BSM — Base of Stolen Memory

B/D/F/Type:	0/2/0/PCI
Address Offset:	5C-5Fh
Default Value:	[Device Specific]
Access:	RO;
Size:	32 bits

Graphics Stolen Memory and TSEG are within DRAM space defined under TOLUD. From the top of low used DRAM, GMCH claims 1 to 64MBs of DRAM for internal graphics if enabled.

Bit	Access	Default Value	Description
31:20	RO	--	**Base of Stolen Memory (BSM):** This register contains bits 31 to 20 of the base address of stolen DRAM memory. The host interface determines the base of graphics stolen memory by subtracting the graphics stolen memory size from TOLUD. See Device 0 TOLUD in the EDS for more explanation.
19:0	RO	00000h	**Reserved**

7.3.22.2 HSRW — Hardware Scratch Read Write

B/D/F/Type: 0/2/0/PCI
Address Offset: 60-61h
Default Value: 0000h
Access: RO;
Size: 16 bits

Bit	Access	Default Value	Description
15:0	RO	0000h	**Reserved**

7.3.22.3 MSAC — Multi Size Aperture Control

B/D/F/Type: 0/2/0/PCI
Address Offset: 62h
Default Value: 02h
Access: RO;
Size: 8 bits

This register determines the size of the graphics memory aperture in function 0 and only in the untrusted space. By default the aperture size is 256 MB. Only the system BIOS will write this register based on pre-boot address allocation efforts, but the graphics may read this register to determine the correct aperture size. System BIOS needs to save this value on boot so that it can reset it correctly during S3 resume.

Bit	Access	Default Value	Description
7:4	RO	0h	**Scratch Bits Only:** Have no physical effect on hardware.
3	RO	0b	**Reserved**
2:1	RO	01b	**Untrusted Aperture Size (LHSAS):** 11: bits [28:27] of GMADR register are made Read only and forced to zero, allowing only 512MB of GMADR 01: bit [28] of GMADR is made R/W and bit [27] of GMADR is forced to zero allowing 256MB of GMADR 00: bits [28:27] of GMADR register are made R/W allowing 128MB of GMADR 10: Illegal programming. ***This bit is LT locked, becomes read-only when trusted environment is launched.***
0	RO	0b	**Reserved**

8 Memory Interface Registers

8.1 Introduction

This chapter describes the memory-mapped registers associated with the Memory Interface, including brief descriptions of their use. The functions performed by these registers are discussed fully in the Memory Interface Functions, Memory Interface Instructions, and Programming Environment chapters.

The registers detailed in this chapter are used across the GenX family of products. However, slight changes may be present in some registers (i.e., for features added or removed), or some registers may be removed entirely. These changes are clearly marked within this chapter.

8.2 Virtual Memory Control

GenX products differ somewhat in the types of virtual memory they support and how they support it. The following table describes the structures to support Global virtual memory (shared between all GFX processes) and per-process virtual memory.

Virtual Memory Structure	All
Global (GGTT)	Anywhere
Per-Process (PPGTT)	Single-level, anywhere

8.2.1 Global Virtual Memory

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

8.2.1.1 PGTBL_CTL—Page Table Control Register

PGTBL_CTL—Page Table Control Register	
Register Type:	MMIO
Address Offset:	2020h
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

The PGTBL_CTL register is used to enable or disable the mapping of graphics memory using the Global Graphics Translation Table (GGTT), set the size, and to set the base (physical) address of the GGTT.

Software must use the following steps to modify the Global GTT directly or update the Global GTT base:

Flush the Gfx Pipeline

Flush the Chipset write buffers using the flush (GFX_FLSH_CTL) register

Update Global GTT using physical address/Update the Global GTT base register

Flush Chipset write buffers using the flush (GFX_FLSH_CTL) register

The GGTT must be 4KByte aligned. The GGTT must reside in un-snooped Main Memory and must be contiguous. The GGTT must be completely contained within physical memory. A memory access that requires fetching a GGTT entry that is not in physical memory will have UNDEFINED results.

[All Devices]: Software can use the GTTADR space to update entries in the GGTT. This allows the device to “snoop” writes to GTTADR and invalidate internal Translation Look-aside Buffers (TLBs) as required.

This register is *not* reset by a graphics reset. It will maintain its value unless a full chipset reset is performed.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:12</td><td>
<table>
<tr><th colspan="2">Page Table Base Address</th></tr>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Address:</td><td>GraphicsAddress[31:12]</td></tr>
<tr><td>Surface Type:</td><td>PageTableEntry</td></tr>
<tr><td colspan="2">This field specifies Bits 31:12 of the starting address of the global GTT.
This address is a physical offset into system memory. This address must be in physical memory, i.e., it must be below the top of memory. Furthermore, the GGTT must be entirely contained within physical memory, i.e., the GTT Size added to the Page Table Base Address must be below top of memory.
This field is only valid when the <b>Page Table Enable</b> field is specified as ENABLED.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th><th>Project</th></tr>
<tr><td>The base address for the GTT is expected to be size aligned in memory. Eg for 512KB size of the GTT bits 18:12 of the address need to be zero</td><td>DevCL</td></tr>
</table>
</td></tr>
</table>

PGTBL_CTL—Page Table Control Register

11:8

Reserved	Project:	All	Format:	MBZ

7:4

Physical Start Address Extension	
Project:	All
Default Value:	0h
Address:	GraphicsAddress[35:32]
This field specifies Bits 35:32 of the starting address of the GGTT.	

3:1

Size of the Global GTT	
Project:	All
Default Value:	0h
Format:	U3

Value	**Name**	**Description**	**Project**
000	512KB	512KB	All
001	256KB	256KB	All
010	128KB	128KB	All
011	1MB	1MB	Reserved
100	2MB	2MB	Reserved
101	1.5MB	1.5MB	Reserved
11X	Reserved	Reserved	All

PGTBL_CTL—Page Table Control Register

0

Page Table Enable

Project:	All
Security:	None
Default Value:	0h
Format:	Enable

This field determines whether GM mappings are enabled. If disabled, GM mapping does not occur except for requests from the CPU and VGA streams. Any accesses to GM (other than CPU read, and VGA streams) while this bit is clear generates a Page Table HW Error (see Page Table Error in *Programming Interface*).

Note: The source of the Page Table HW Error is available only via the *Debug* PGTBL_ER register.

Value	**Name**	**Description**	**Project**
0h	Disable	GM mapping does not occur except for requests from the CPU and VGA streams.	All
1h	Enable	ENABLED	All

8.2.1.2 PGTBL_ER—Page Table Error Register *(Debug)*

PGTBL_ER—Page Table Error Register

Register Type:	MMIO
Address Offset:	2024h
Project:	All
Default Value:	0000 0000h
Access:	RO
Size (in bits):	32

This register applies when the Per-Process Virtual Address Space and Run List Enable is clear else see below

The PGTBL_ER *Debug* register stores information indicating the source of an error associated with GM mapping via the GTT. Note that this is a READ-ONLY register and cannot be modified by software.

Error Types:

XX _INVALID_GTT_PTE: Translated Page Table Entry (PTE) is marked as not valid. Implemented by all streams. Detected at translation time for either Global or Per-Process GTT.

XX _INVALID_PTE_DATA: The PTE was marked valid, though the memory space or page mapped is not considered legal (i.e., Address points to PAM, SMM, over TOM and other restricted spaces in Main Memory). Implemented by Host Only.

CS_INVALID_GTT: Set if a ring buffer is active and the Page table is not enabled.

This register identifies the TLB that detected the error. After an error, Normal priority data streams Commands, Render Cache and Mapping Cache stop execution. GTT errors on Host reads are not recorded. If there is an error on a read access a read request is forwarded to a memory address and data obtained from memory is returned to the CPU. Errors on Host writes are recorded and the write is completed with byte enables off.

Each Source records the first error and ignores subsequent errors.

Bit	Description
31:27	**Reserved** Project: All Format: MBZ
26	**MT_INVALID_GTT_PTE** Project: All Format: Flag Invalid Sampler Cache GTT entry
25	**Reserved** Project: All Format: MBZ

PGTBL_ER—Page Table Error Register

<table>
<tr><td>24</td><td>
<table>
<tr><td colspan="2"><b>LC_INVALID_GTT_PTE</b></td></tr>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>Flag</td></tr>
<tr><td colspan="2">Invalid Render Cache GTT entry</td></tr>
</table>
<table>
<tr><th>Errata</th><th>Description</th><th>Project</th></tr>
<tr><td>GENX016</td><td>This bit will never be set.</td><td>All</td></tr>
</table>
</td></tr>
<tr><td>23

22</td><td>
<table>
<tr><td><b>ISC_INVALID_GTT_PTE</b></td><td>Project:</td><td>All</td><td>Format:</td><td>Flag</td></tr>
<tr><td colspan="5">Invalid Instruction/State Cache GTT entry</td></tr>
</table>
<table>
<tr><td><b>ROC_INVALID_GTT_PTE</b></td><td>Project:</td><td>All</td><td>Format:</td><td>Flag</td></tr>
<tr><td colspan="5">Reserved since there is no ROC</td></tr>
</table>
</td></tr>
<tr><td>21</td><td>
<table>
<tr><td><b>CS_VertexData_INVALID_GTT_PTE</b></td><td>Project:</td><td>All</td><td>Format:</td><td>Flag</td></tr>
<tr><td colspan="5">Invalid GTT Entry during Vertex Fetch</td></tr>
</table>
</td></tr>
<tr><td>20</td><td>
<table>
<tr><td><b>CS_Command_INVALID_GTT_PTE</b></td><td>Project:</td><td>All</td><td>Format:</td><td>Flag</td></tr>
<tr><td colspan="5">Invalid GTT Entry during Command Fetch</td></tr>
</table>
</td></tr>
<tr><td>19</td><td>
<table>
<tr><td><b>CS_INVALID_GTT</b></td><td>Project:</td><td>All</td><td>Format:</td><td>Flag</td></tr>
<tr><td colspan="5"></td></tr>
</table>
</td></tr>
<tr><td>18</td><td>
<table>
<tr><td><b>CRSR _INVALID_GTT_PTE</b></td><td>Project:</td><td>All</td><td>Format:</td><td>Flag</td></tr>
<tr><td colspan="5">Invalid GTT Entry during Cursor Fetch</td></tr>
</table>
</td></tr>
<tr><td>17</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>16</td><td>
<table>
<tr><td colspan="2"><b>OVRL_INVALID_GTT_PTE</b></td></tr>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>Flag</td></tr>
<tr><td colspan="2">Invalid GTT Entry during Overlay Fetch</td></tr>
</table>
<table>
<tr><th>Errata</th><th>Description</th><th>Project</th></tr>
<tr><td>BWT010</td><td>Invalid GTT Entry during Overlay Fetch is ignored.</td><td>DevBW-A, DevBW-B</td></tr>
</table>
</td></tr>
</table>

PGTBL_ER—Page Table Error Register

15:13	**Reserved**	Project:	All	Format :	MBZ
12	**DISPC_INVALID_GTT_PTE**	Project:	All	Format:	Flag
	Invalid GTT Entry during Display C Fetch				
11:9	**Reserved**	Project:	All	Format :	MBZ
8	**DISPB_INVALID_GTT_PTE**	Project:	All	Format:	Flag
	Invalid GTT Entry during Display B Fetch				
7:5	**Reserved**	Project:	All	Format :	MBZ
4	**DISPA_INVALID_GTT_PTE**	Project:	All	Format:	Flag
	Invalid GTT Entry during Display A Fetch				
3:2	**Reserved**	Project:	All	Format :	MBZ
1	**HOST_ INVALID_PTE_DATA**	Project:	All	Format:	Flag
	Valid PTE references illegal memory, such as PAM, SMM or TOM				
0	**HOST_INVALID_GTT_PTE**				
	Project:	All			
	Default Value:	0h			
	Format:	Flag			
	Invalid GTT Entry during Fetch on behalf of the Host				

Errata	**Description**	**Project**
BWT015	This bit will never be set.	DevBW

8.2.1.3 PGTBL_ER—Page Table Error Register *(Debug)* [Per-Process GTT enabled on CTG]

PGTBL_ER—Page Table Error Register

Register Type:	MMIO
Address Offset:	2024h
Project:	DevCTG
Default Value:	00000000h
Access:	RO
Size (in bits):	32

The PGTBL_ER *Debug* register stores information indicating the source of an error associated with GM mapping via a the PPGTT Page Directory (see

Two-Level Per-Process Virtual Memory ([DevCTG] Only)). Note that this is a READ-ONLY register and cannot be modified by software.

This register identifies the TLB that detected the error. After an error, Normal priority data streams stop execution.

Only the first error is recorded; subsequent errors are ignored.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:27</td><td>
<table>
<tr><td colspan="2"><b>Invalid GTT Client Identifier</b></td></tr>
<tr><td>Project:</td><td>pre-HVN/ABD</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>U5</td></tr>
<tr><td colspan="2">For Bit Stream</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>When Bit 3 = 0</td><td>Normal Priority Clients</td><td>Normal Priority Clients</td><td>DevCTG</td></tr>
<tr><td>11101</td><td>Command Stream TLB</td><td>Command Stream TLB</td><td>DevCTG</td></tr>
<tr><td>11110</td><td>Data TLB</td><td>Data TLB</td><td>DevCTG</td></tr>
<tr><td>11111</td><td>Scratch TLB</td><td>Scratch TLB</td><td>DevCTG</td></tr>
</table>
</td></tr>
<tr><td>26</td><td>
<table>
<tr><td><b>Invalid GTT or PD Error</b></td><td>Project:</td><td>DevCTG</td><td>Format:</td><td>Flag</td></tr>
<tr><td colspan="5">Error code can be read in Invalid GTT Error field. In the Bit Stream pipeline</td></tr>
</table>
</td></tr>
</table>

PGTBL_ER—Page Table Error Register

25:21

Invalid GTT Client Identifier	
Project:	DevCTG
Default Value:	0h
Format:	U5
This field is only valid if Invalid GTT or PD Error is set.	

Value	Name	Description	Project
When Bit 3 = 0	Normal Priority Clients	Normal Priority Clients	DevCTG
00001	Command Stream	Command Stream	DevCTG
00011	Vertex Fetch	Vertex Fetch	DevCTG
00101	Texture Cache (MT)	Texture Cache (MT)	DevCTG
01001	Render Cache Color (RCC)	Render Cache Color (RCC)	DevCTG
01011	Instruction/State Cache (ISC)	Instruction/State Cache (ISC)	DevCTG
11001	Render Cache Depth (RCZ)	Render Cache Depth (RCZ)	DevCTG
10110	CS PD Read	CS PD Read	DevCTG

20

Invalid GTT or PD Error	Project:	DevCTG	Format:	Flag
Error code can be read in Invalid GTT Error field				

19

CS_INVALID_GTT	Project:	DevCTG	Format:	Flag

18

CRSR _INVALID_GTT_PTE	Project:	DevCTG	Format:	Flag
Invalid GTT Entry during Cursor Fetch				

17:9

Reserved	Project:	DevCTG	Format:	MBZ

8

DISPB_INVALID_GTT_PTE	Project:	DevCTG	Format:	Flag
Invalid GTT Entry during Display B Fetch				

7:5

Reserved	Project:	DevCTG	Format:	MBZ

4

DISPA_INVALID_GTT_PTE	Project:	DevCTG	Format:	Flag
Invalid GTT Entry during Display A Fetch				

3:2

Reserved	Project:	DevCTG	Format:	MBZ

PGTBL_ER—Page Table Error Register

1	**HOST_ INVALID_PTE_DATA**	Project:	DevCTG	Format:	Flag
	Valid PTE references illegal memory, such as PAM, SMM or TOM				
0	**HOST_INVALID_GTT_PTE**	Project:	DevCTG	Format:	Flag
	Invalid GTT Entry during Fetch on behalf of the Host				

8.2.2 PGTBL_ER—Page Table Error Register *(Debug)* [Per-Process GTT enabled on CTG]

PGTBL_ER—Page Table Error Register

Register Type:	MMIO
Address Offset:	4024h
Project:	DevGT+
Default Value:	00000000h
Access:	RO
Size (in bits):	32

The PGTBL_ER *Debug* register stores information indicating the source of an error associated with GM mapping via a the PPGTT Page Directory (see

Two-Level Per-Process Virtual Memory ([DevCTG] Only)). Note that this is a READ-ONLY register and cannot be modified by software.

This register identifies the TLB that detected the error. After an error, Normal priority data streams stop execution.

Only the first error is recorded; subsequent errors are ignored.

Bit	Description
31:27	**Invalid GTT Client Identifier** Project: DevCTG Default Value: 0h Format: U5 For Bit Stream
26	**Invalid GTT or PD Error** Project: DevCTG Format: Flag Error code can be read in Invalid GTT Error field. In the Bit Stream pipeline

Bits 31:27 values:

Value	Name	Description	Project
When Bit 3 = 0	Normal Priority Clients	Normal Priority Clients	DevCTG
11101	Command Stream TLB	Command Stream TLB	DevCTG
11110	Data TLB	Data TLB	DevCTG
11111	Scratch TLB	Scratch TLB	DevCTG

PGTBL_ER—Page Table Error Register

25:21

Invalid GTT Client Identifier	
Project:	DevCTG
Default Value:	0h
Format:	U5
This field is only valid if Invalid GTT or PD Error is set.	

Value	**Name**	**Description**	**Project**
When Bit 3 = 0	Normal Priority Clients	Normal Priority Clients	DevCTG
00001	Command Stream	Command Stream	DevCTG
00011	Vertex Fetch	Vertex Fetch	DevCTG
00101	Texture Cache (MT)	Texture Cache (MT)	DevCTG
01001	Render Cache Color (RCC)	Render Cache Color (RCC)	DevCTG
01011	Instruction/State Cache (ISC)	Instruction/State Cache (ISC)	DevCTG
11001	Render Cache Depth (RCZ)	Render Cache Depth (RCZ)	DevCTG
10110	CS PD Read	CS PD Read	DevCTG

20	**Invalid GTT or PD Error**	Project:	DevCTG	Format:	Flag
	Error code can be read in Invalid GTT Error field				
19	**CS_INVALID_GTT**	Project:	DevCTG	Format:	Flag
18	**CRSR _INVALID_GTT_PTE**	Project:	DevCTG	Format:	Flag
	Invalid GTT Entry during Cursor Fetch				
17:9	**Reserved**	Project:	DevCTG	Format:	MBZ
8	**DISPB_INVALID_GTT_PTE**	Project:	DevCTG	Format:	Flag
	Invalid GTT Entry during Display B Fetch				
7:5	**Reserved**	Project:	DevCTG	Format:	MBZ
4	**DISPA_INVALID_GTT_PTE**	Project:	DevCTG	Format:	Flag
	Invalid GTT Entry during Display A Fetch				
3:2	**Reserved**	Project:	DevCTG	Format:	MBZ
1	**HOST_ INVALID_PTE_DATA**	Project:	DevCTG	Format:	Flag
	Valid PTE references illegal memory, such as PAM, SMM or TOM				
0	**HOST_INVALID_GTT_PTE**	Project:	DevCTG	Format:	Flag
	Invalid GTT Entry during Fetch on behalf of the Host				

8.2.2.1 Graphics Translation Table (GTT) Range (GTTADR)

Address Offset:	GTTADR in CPU Physical Space
Access:	Aligned DWord Read/Write
The GTTADR memory BAR defined in graphics device config space is an alias for the Global GTT.	
Programming Notes: It is recommended that the driver map all graphics memory pages in the GGTT to some physical page, if only a dummy page.	

8.2.2.2 GTT Page Table Entries (PTEs)

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

31 12	11:8	7:4	3	2 1	0
Physical Page Address 31:12	Reserved:MBZ	Physical Page Address 35:32	Reserved	Mapping Type	Valid

Bits	Description
31: 12	**Physical Page Address 31:12:** If the Valid bit is set, This field provides the page number of the physical memory page backing the corresponding Graphics Memory page.
11:8	Reserved: MBZ
7:4	Physical Start Address Extension: This field specified Bits 35:32 of the page table entry. This field must be zero for 32 bit addresses.
3	Reserved: MBZ
2:1	**Mapping Type:** If the Valid bit is set, this field specifies the type of physical memory backing this Graphics Memory page, as defined below: 0: Physical address targets Main Memory (not snooped). Physical address is a main memory page number (including pages in stolen memory). 1-2: Reserved 3: Physical address targets cacheable Main Memory (aka System Memory) (causes snoop on processor bus). Must not be targeted by the processor through graphics memory range. Accesses via the Instruction stream are permitted (no error generated), yet treated as unsnooped Main Memory. This removes restrictions regarding Instruction stream overfetches into dissimilar graphics memory regions.
0	**Valid PTE:** This field indicates whether the mapping of the corresponding Graphics Memory page is valid. 1: Valid 0: Invalid. An access (other than a CPU Read) through an invalid PTE will result in Page Table Error (Invalid PTE).

8.2.3 Single-Level (Flat) Per-Process Virtual Memory

8.2.3.1 PGTBL_CTL2— Per Process Page Table Control Register

PGTBL_CTL2— Per Process Page Table Control Register

Register Type:	MMIO
Address Offset:	20C4h
Project:	DevBW, DevCL
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

The PGTBL_CTL2 register is used to enable the secondary mapping of graphics memory addresses by defining the starting point of the per-process Graphics Translation Table (PPGTT).

Software must assure that a pipeline flush occurs subsequent to updating any PPGTT entries or changing the value of the Page Table Base Address and prior to any new access in the PPGTT aperture.

Once a PPGTT is established, software can update entries of the PPGTT using physical writes. The PPGTT does not have an access window corresponding to GTTADR that will trigger snoops and/or flushes when possibly pre-fetched entries are modified.

The PPGTT can be up to 1MB in size as programmed below. Each 4B entry in the PPGTT corresponds to a 4KB page of memory mapped through the PPGTT aperture.

The PPGTT must be 4KByte-aligned. The PPGTT must reside in unsnooped Main Memory and must be contiguously size aligned. This register is saved and restored per context. If the valid bit for this register is not set, the hardware uses the Global GTT.

Bit	Description
31:12	**Page Table Base Address** Project: All Default Value: 0h Address: GraphicsAddress[31:12] Surface Type: PageTableEntry This field specifies Bits 31:12 of the starting address of the GTT. Bit 1 of the address is MBZ. This address is a physical offset into system memory. This field is only valid when the **Page Table Enable** field is specified as ENABLED. Format = "Effective Local Memory Address" Bits 31:2
11:8	**Reserved** Project: All Format: MBZ

PGTBL_CTL2— Per Process Page Table Control Register

7:4

Physical Start Address Extension	
Project:	All
Default Value:	0h
Address:	GraphicsAddress[35:2]

This field specified Bits 35:32 of the page table entry. This field must be zero for 32 bit addresses.

3:1

Size of the PPGTT	
Project:	All
Default Value:	0h
Format:	U3

Value	Name	Description	Project
000	64KB	64KB	All
001	128KB	128KB	All
010	256KB	256KB	All
011	512KB	512KB	All
100	1MB	1MB	All
101-111	Reserved	Reserved	All

0

Page Table Enable	
Project:	All
Default Value:	0h
Format:	Enable

This field determines whether GM mappings are enabled. If enabled, the Page Table Base Address specifies the starting address of the PGTT. If disabled, GM mapping will proceed using the global GTT.

Value	Name	Description	Project
0h	Disable	GM mapping will proceed using the global GTT.	All
1h	Enable	The Page Table Base Address specifies the starting address of the PGTT	All

8.2.3.2 PGTBL_STR2—Page Table Steer Register (Per Process)

PGTBL_STR2—Page Table Steer Register (Per Process)	
Register Type:	MMIO
Address Offset:	20C8h
Project:	DevBW, DevCL
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

The PGTBL_STR2 register is used to map the graphics functions to either the per-process GTT or the global GTT.

This register is saved and restored with context. If the valid bit for the per-process GTT is not set, the hardware uses the Global GTT for all functions and ignores the contents of this register.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:22</td><td><table><tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>21:16</td><td><table><tr><td><b>Write enable bits</b></td><td>Project:</td><td>All</td><td>Format:</td><td>Mask[5:0]</td></tr><tr><td colspan="5">This bit needs to be set in order to change the value for the corresponding location of register bits 5:0</td></tr></table></td></tr>
<tr><td>15:6</td><td><table><tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>5</td><td><table><tr><td colspan="2"><b>Location of the render batch buffer</b></td></tr><tr><td>Project:</td><td>All</td></tr><tr><td>Default Value:</td><td>0h</td></tr><tr><td>Format:</td><td>U1</td></tr><tr><td colspan="2">Location of the render batch buffer</td></tr></table>
<table><tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr><tr><td>0h</td><td></td><td>Batch buffer accesses are translated through the global GTT</td><td>All</td></tr><tr><td>1h</td><td></td><td>Batch buffer accesses are translated through the per-process GTT (PGTT).</td><td>All</td></tr></table></td></tr>
</table>

PGTBL_STR2—Page Table Steer Register (Per Process)

4

Location of indirect state buffers includes states and instructions	
Project:	All
Default Value:	0h
Format:	U1
Location of indirect state buffers includes states and instructions	

Value	Name	Description	Project
0h		Indirect state buffer accesses are translated through the global GTT	All
1h		Indirect state buffer accesses are translated through the per-process GTT (PGTT).	All

3

Location of Vertex buffer	
Project:	All
Default Value:	0h
Format:	U1
Location of Vertex buffer	

Value	Name	Description	Project
0h		Vertex buffer accesses are translated through the global GTT	All
1h		Vertex buffer accesses are translated through the per-process GTT (PGTT).	All

2

Reserved	Project:	All	Format:	MBZ

1

Location of functions using the Sampler cache	
Project:	All
Default Value:	0h
Format:	U1
Location of functions using the Sampler cache	

Value	Name	Description	Project
0h		Sampler surface accesses are translated through the global GTT	All
1h		Sampler surface accesses are translated through the per-process GTT (PGTT).	All

PGTBL_STR2—Page Table Steer Register (Per Process)

0

Location of functions using the render cache

Project:	All
Default Value:	0h
Format:	U1

Includes Render targets, constants, Scratch Space access and direct reads/writes from EUs to memory.

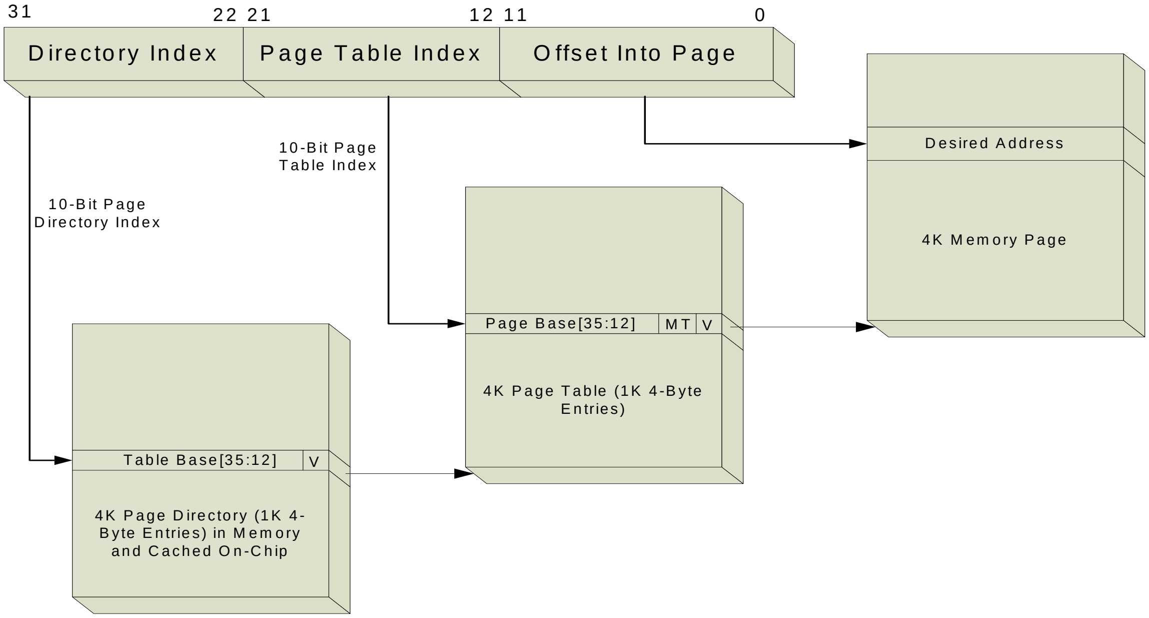
Value	Name	Description	Project
0h		Render surface accesses are translated through the global GTT	All
1h		Render surface accesses are translated through the per-process GTT (PGTT).	All

8.2.4 Two-Level Per-Process Virtual Memory ([DevCTG] Only)

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

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

The PPGTT is disabled by resetting the **Per-Process Virtual Address Space and Run List Enable** bit in the GFX_MODE – Graphics Mode Register. Run Lists are described later in this chapter.



PPGTT Lookup

8.2.4.1 PPGTT Table Entries (PTEs)

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

31 12	11:8	7:4	3	2 1	0
Physical Page Address 31:12	Reserved:MBZ	Physical Page Address 35:32	Reserved	Mapping Type	Valid

Bits	Description
31:12	**Physical Page Address 31:12**: If the Valid bit is set, This field provides the page number of the physical memory page backing the corresponding Graphics Memory page.
11:8	Reserved: MBZ
7:4	**Physical Page Address Extension:** This field specifies bits 35:32 of the directory entry.
3:1	Reserved: MBZ
2:1	**Mapping Type:** If the Valid bit is set, this field specifies the type of physical memory backing this Graphics Memory page, as defined below: 0: Physical address targets Main Memory (not snooped). Physical address is a main memory page number (including pages in stolen memory). 1-2: Reserved 3: Physical address targets cacheable Main Memory (aka System Memory) (causes snoop on processor bus). Must not be targeted by the processor through graphics memory range, such accesses will not be snooped after translation. Accesses via the Instruction stream are permitted (no error generated), yet treated as unsnooped Main Memory. This removes restrictions regarding Instruction stream overfetches into dissimilar graphics memory regions.
0	**Valid PTE:** This field indicates whether the mapping of the corresponding Graphics Memory page is valid. 1: Valid 0: Invalid. An access (other than a CPU Read) through an invalid PTE will result in Page Table Error (Invalid PTE).

8.2.4.2 PP_DIR_BASE – Page Directory Base Register

PP_DIR_BASE – Page Directory Base Register	
Register Type:	MMIO
Address Offset:	2518h
Project:	DevCTG
Default Value:	0000 0000h
Access:	R/W (Debug Only)
Size (in bits):	32

This register contains the offset into the GGTT where the (current context's) PPGTT page directory begins. This register is restored with context. SW should not normally modify this register. The Page Directory Base Address is set by SW only by modifying the value of this register in the context image such that the new value is restored the next time the context runs.

Bit	Description
30:16	**Page Directory Base Offset** Project: DevCTG Default Value: 0h Format: U15 Range [0,GGTT Size in cachelines - 1] Contains the cacheline (64-byte) offset into the GGTT where the page directory begins.
15:0	**Reserved** Project: DevCTG Format: MBZ

8.2.4.3 PP_DCIR – PPGTT Directory Cache Index Register

PP_DCIR – PPGTT Directory Cache Index Register

Register Type:	MMIO
Address Offset:	2500h
Project:	DevCTG
Default Value:	0000 0000h
Access:	R/W (Debug Only)
Size (in bits):	32

The Directory Cache Index and Data Registers are used to read the contents of the on-chip directory cache. This on-chip cache is updated only when the context is restored (all entries that are enabled in the **PPGTT Directory Cache Valid Registers**).

This register contains the DW offset into the on-chip PPGTT directory cache. A read to the PPGTT Directory Cache data register will return the entry at the DW indicated by this index.

Bit	Description
31:12	**Reserved** | Project: | DevCTG | Format: | MBZ
11:2	**PPGTT Directory Cache Index** Project: DevCTG Default Value: 0h Format: U10 Range [0,1023] Contains the DW offset into the on-chip directory cache that will be read through the PPGTT Directory Cache Data Register.
1:0	**Reserved** | Project: | DevCTG | Format: | MBZ

8.2.4.4 PP_DCDR – PPGTT Directory Cache Data Register

PP_DCDR – PPGTT Directory Cache Data Register

Register Type:	MMIO
Address Offset:	2504h
Project:	DevCTG
Default Value:	0000 0000h
Access:	RO (Debug Only)
Size (in bits):	32

This register returns the on-chip PPGTT Directory Cache entry indicated by the PPGTT Directory Cache Index. The PPGTT Directory Cache can contain HPAs if VT-d is enabled; bit 3 indicates if this entry is an HPA. The format of this register matches PPGTT Directory Entries (PDEs) except for bit 3.

Bit	Description
31:12	**Physical Page Address 31:12** Project: DevCTG Default Value: 0h Address: PhysicalAddress[31:12] If the Valid bit is set, This field provides the page number of the physical memory page backing the corresponding Graphics Memory page
11:8	**Reserved** Project: All Format: MBZ
7:4	**Physical Page Address Extension** Project: DevCTG Default Value: 0h Address: GraphicsAddress[35:32] This field specifies bits 35:32 of the directory entry
3	**Physical Address is Host Physical Address** Project: All Format: ??? This bit indicates that this entry is an HPA. This bit can only be set when VT-d is enabled. If clear, this entry is a GPA. This bit will always be clear when VT-d is not enabled. This bit should be written as 0.
2:1	**Reserved** Project: All Format: MBZ

PP_DCDR – PPGTT Directory Cache Data Register

0	**Valid PDE** Project: DevCTG Default Value: 0h Format: Flag This field indicates whether this directory entry is valid.

Value	Name	Description	Project
0h	Invalid	An access through an invalid PDE will result in a fatal error (hang).	All
1h	Valid		All

8.2.4.5 PP_DCLV – PPGTT Directory Cacheline Valid Register

PP_DCLV – PPGTT Directory Cacheline Valid Register

Register Type:	MMIO
Address Offset:	2508h
Project:	DevCTG
Default Value:	FFFF FFFFh; FFFF FFFFh
Access:	RO
Size (in bits):	64

This register controls update of the on-chip PPGTT Directory Cache during a context restore. Bits that are set will trigger the load of the corresponding 16 directory entry group. This register is restored with context (prior to restoring the on-chip directory cache itself). This register is also restored when switching to a context whose LRCA matches the current CCID if the **Force PD Restore** bit is set in the context descriptor.

The context image of this register must be updated and maintained by SW; SW should not normally need to read this register.

This register can also effectively be used to limit the size of a processes' virtual address space. Any access by a process that requires a PD entry in a set that is not enabled in this register will cause a fatal error, and no fetch of the PD entry will be attempted

Bit	Description
63:32	**Reserved** Project: DevCTG Format: MBZ
31:0	**PPGTT Directory Cache Restore [1..32] 16 entries** Project: DevCTG Format: Array:Enable If set, the [1st..32nd] 16 entries of the directory cache are considered valid and will be brought in on context restore. If clear, these entries are considered invalid and fetch of these entries will not be attempted.

8.2.5 PPGTT Page Fault Interface ([DevCTG] Only)

This interface is used to signal page faults that occur during access of per-process virtual graphics memory. A fault of this nature will stall the 3D/Media pipeline behind the fault, and all new TLB requests from anywhere in the pipeline will be stalled. Faults are recorded in a fault log consisting of 32 fault slots.

When a TLB request (the first access to a page that is not present in a client's TLB cache) results in a fault, any further TLB requests (from any client) will be stalled. Currently pending TLB requests will be completed (it is possible that some of these will fault as well). Once no more TLB requests are outstanding, a page fault interrupt will be sent. When reading the Page Fault Indication Register after receiving the Page Fault interrupt, SW is assured that no more faults will be logged while it is in the process of reading the page fault registers. When all faults have been serviced, SW should clear the Fault Indication Register and then clear the **Fault In Service** bit of the PPGTT Page Fault Interface Control Register.

8.2.5.1 PP_PFIR – PPGTT Page Fault Indication Register

PP_PFIR – PPGTT Page Fault Indication Register

Register Type:	MMIO
Address Offset:	2510h
Project:	DevCTG
Default Value:	0000 0000h
Access:	R/WC
Size (in bits):	32

This register contains the flags for page faults. All bits should be cleared at once by writing FFFFFFFFh to this register once all faults have been serviced. No additional bits of this register will become set (signaling additional faults) between the time the page fault interrupt has been sent to the host and the time the host clears the **Fault In Service** bit indicating it is done servicing faults

Bit	Description
31:0	**Page Fault [31:0]** | Project: DevCTG | Format: Array:Flag Fault indicator for page fault log index [31:0]. When set, this flag indicates that a page fault is outstanding. The invalid page address that was accessed can be read from fault entry [31:0]. SW should clear this bit by writing a '1' to it to indicate to HW that the fault has been serviced (the page has been mapped and should now be valid).

8.2.5.2 PP_PFIC – PPGTT Page Fault Interface Control

PP_PFIC – PPGTT Page Fault Interface Control

Register Type:	MMIO
Address Offset:	2514h
Project:	DevCTG
Default Value:	00000000h
Access:	R/WC
Size (in bits):	32

This register contains a bit to allow SW to synchronize with HW in order to read faults out of the fault log.

Bit	Description
31	**Fault In Service** | Project: DevCTG | Format: Flag This R/WC bit is set by HW to indicate that a fault has been signaled via interrupt to SW. SW should clear this bit by writing a '1' to it once it is done servicing faults so that HW can resume normal operation. Any bits still set in the Page Fault Indication Register when this bit is cleared will trigger a new interrupt. Writing a '0' to this bit has no effect.
30:0	**Reserved** | Project: DevCTG | Format: MBZ Write as 0's.

8.2.5.3 PP_PFD[0:31] – PPGTT Page Fault Data Registers

PP_PFD[0:31] – PPGTT Page Fault Data Registers	
Register Type:	MMIO
Address Offset:	2580h
Project:	DevCTG
Default Value:	0000 0000h
Access:	RO
Size (in bits):	32

The GTT Page Fault Log entries can be read from these registers.

2580h-2583h: Fault Entry 0
…

25FCh-25FFh: Fault Entry 31

DWord	Bit	Description
0..31	31:12	**Fault Entry Page Address** Project: All Address: GraphicsAddress[31:12] This RO field contains the faulting page address for this Fault Log entry. This field will contain a valid fault address only if the bit in the GTT Page Fault Indication Register corresponding with the address offset of this entry is set.
	11:0	**Reserved** Project: All Format: MBZ

8.2.6 TLB Read Interface

It may be necessary for one or more pages belonging to a context to be unmapped from its PPGTT in order to map other pages when resolving a page fault. Pages that get unmapped cannot be one of the set that the HW is currently using. SW should read all of the TLB entry virtual addresses in order to report these virtual page addresses to the OS/Scheduler such that it can avoid swapping these pages out in order to bring in a page to resolve a fault.

8.2.6.1 TLB_RD_EXT — TLB Read Extent

TLB_RD_EXT -- TLB Read Extent	
Register Type:	MMIO
Address Offset:	251Ch
Project:	All
Default Value:	0000 0780h
Access:	RO
Size (in bits):	32
This RO register can be read by software to determine how many TLB Read entries follow. SW must read the entire set to make sure all in-use pages are reported during the servicing of a page fault.	

Bit	Description
31:2	**TLB Read Extent** Project: All Default Value: 01E0h ??? Format: U30 This RO register is hardwired to a count of the total number of TLB read registers SW can read this register to determine the total range of potentially valid DWs in the TLB read range.
1:0	**Reserved** Project: All Format: MBZ

Unused registers in the range below, from B000h to B000h + TLB Read Extent, should be treated as reserved and read as 0. This allows SW to read the entire range contiguously and maintain proper behavior when reading unused, reserved registers.

8.2.6.2 Instruction/State Cache (ISC)

Instruction/State Cache (ISC)

Register Type:	MMIO
Address Offset:	B000h
Project:	All
Default Value:	TBD
Access:	RO
Size (in bits):	16x32

DWord	Bit	Description
0..15	31:12	**TLB Page Address** Project: All Address: GraphicsVirtualAddress[31:12] If the Valid bit is set, this field contains the page address of the TLB entry.
	11:2	**Reserved** Project: All Format: MBZ
	1	**Global GTT Address** Project: All Format: MBZ Hardwired to 0.
	0	**Valid** Project: All Format: Enable If this bit is set, this entry contains a valid TLB entry. If clear, this TLB entry is effectively “empty” and may be disregarded.

8.2.6.3 Vertex Fetch (VF)

Vertex Fetch (VF)	
Register Type:	MMIO
Address Offset:	B100h
Project:	All
Default Value:	TBD
Access:	RO
Size (in bits):	19x32

DWord	Bit	Description
0..18	31:12	**TLB Page Address** Project: All Address: GraphicsVirtualAddress[31:12] If the Valid bit is set, this field contains the page address of the TLB entry.
	11:2	**Reserved** Project: All Format: MBZ
	1	**Global GTT Address** Project: All Format: MBZ Hardwired to 0.
	0	**Valid** Project: All Format: Enable If this bit is set, this entry contains a valid TLB entry. If clear, this TLB entry is effectively “empty” and may be disregarded.

8.2.6.4 Command Streamer (CS)

Command Streamer (CS)	
Register Type:	MMIO
Address Offset:	B200h
Project:	All
Default Value:	TBD
Access:	RO
Size (in bits):	6x32

DWord	Bit	Description
0..5	31:12	**TLB Page Address** Project: All Address: GraphicsVirtualAddress[31:12] If the Valid bit is set, this field contains the page address of the TLB entry.
	11:2	**Reserved** Project : All Format : MBZ
	1	**Global GTT Address** Project: All Format: Flag If set, this virtual address is a global GTT address, and is guaranteed to remain mapped. Only TLB entries with this bit clear need to be communicated as being part of a minimum set that must remain mapped during the servicing of a page fault. Only the Command Streamer (CS) may contain global GTT entries in its TLB; all the other clients will hardwire this bit to 0 in all of their TLB read registers.
	0	**Valid** Project: All Format: Enable If this bit is set, this entry contains a valid TLB entry. If clear, this TLB entry is effectively “empty” and may be disregarded.

8.2.6.5 Texture Cache (MT)

Texture Cache (MT)	
Register Type:	MMIO
Address Offset:	B300h
Project:	All
Default Value:	???
Access:	RO
Size (in bits):	32x32

DWord	Bit	Description
0..31	31:12	**TLB Page Address** Project: All Address: GraphicsVirtualAddress[31:12] If the Valid bit is set, this field contains the page address of the TLB entry.
	11:2	**Reserved** Project: All Format: MBZ
	1	**Global GTT Address** Project: All Format: MBZ Hardwired to 0.
	0	**Valid** Project: All Format: Enable If this bit is set, this entry contains a valid TLB entry. If clear, this TLB entry is effectively “empty” and may be disregarded.

8.2.6.6 Render Cache (RC)

Render Cache (RC)	
Register Type:	MMIO
Address Offset:	B400h
Project:	All
Default Value:	???
Access:	RO
Size (in bits):	224x32

224 DWords

DWord	Bit	Description
0.223	31:12	**TLB Page Address** Project: All Address: GraphicsVirtualAddress[31:12] If the Valid bit is set, this field contains the page address of the TLB entry.
	11:2	**Reserved** Project: All Format: MBZ
	1	**Global GTT Address** Project: All Format: MBZ Hardwired to 0.
	0	**Valid** Project: All Format: Enable If this bit is set, this entry contains a valid TLB entry. If clear, this TLB entry is effectively “empty” and may be disregarded.

8.3 GFX_MODE – Graphics Mode Register

GFX_MODE – Graphics Mode Register	
Register Type:	MMIO
Address Offset:	2520h
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Trusted Type:	1

This register contains a control bit for the new run list and 2-level PPGTT functions. This register is not saved/restored with context. This register is not reset with single-engine GFX reset; it is only reset by a global graphics reset (all engines including display).

Bit	Description
31:16	**Mask Bits** Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
15	**Reserved** Project: All Format: MBZ
14	**Reserved** Project: All Format: MBZ
13	**Reserved** Project: All Format: MBZ
12:0	**Reserved** Project: All Format: MBZ

8.4 EXCC—Execute Condition Code Register

EXCC—Execute Condition Code Register

Register Type:	MMIO
Address Offset:	2028h
Project:	All
Default Value:	00000000h
Access:	R/W,RO
Size (in bits):	32
Trusted Type:	1

This register contains user defined and hardware generated conditions that are used by MI_WAIT_FOR_EVENT commands. An MI_WAIT_FOR_EVENT instruction excludes the executing ring from arbitration if the selected event evaluates to a "1", while instruction is discarded if the condition evaluates to a "0". Once excluded a ring is enabled into arbitration when the selected condition evaluates to a "0".

Bit	Description
31:22	**Reserved** Project: All Format: MBZ
21	**Mask Bits** Format: Mask[5] This bit serves as a write enable for bit 5. If this register is written with this bit clear the corresponding bit in the field 5 will not be modified. Reading these bits always returns 0s.
20:16	**Mask Bits** Format: Mask[4:0] These bits serves as a write enable for bits 4:0. If this register is written with any of these bits clear the corresponding bit in the field 4:0 will not be modified. Reading these bits always returns 0s.
15:12	**Reserved** Project: All Format: MBZ
11	**Pending Indirect State Dirty Bit** Project: All Format: U32 This field keeps track of whether or not an indirect state pointer command has been parsed in the current context. Clears either on a context save or explicitly through a flush command
10:8	**Pending Indirect State Counter** This field keeps track of the maximum number of indirect state pointers pending in the system. When the register is saved/restored, it saves either a value of 1 or 0. This field is Read-Only
7:6	**Reserved** Project: All Format: MBZ

EXCC—Execute Condition Code Register

<table>
<tr><td>5</td><td>
<table>
<tr><th>Indirect State Pointer Force Restore</th></tr>
<tr><td>Determines whether to use pending indirect state counter to restore data to memory, or restore indirect data</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>Use Pending</td><td>Use the pending indirect state counter to restore data to memory</td><td>All</td></tr>
<tr><td>1h</td><td>Don’t Use Pending</td><td>Don’t use pending indirect state counter to restore data to memory. Always restore indirect data</td><td>All</td></tr>
</table>
</td></tr>
<tr><td>4:0</td><td>
<table>
<tr><th>User Defined Condition Codes</th></tr>
<tr><td>The software may signal a Stream Semaphore by setting the Mask bit and Signal Bit together to match the bit field specified in a WAIT_FOR_EVENT (Semaphore).</td></tr>
</table>
</td></tr>
</table>

8.5 RINGBUF—Ring Buffer Registers

RING_BUFFER_TAIL

Register Type:	MMIO
Address Offset:	2030h
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

These registers are used to define and operate the "ring buffer" mechanism which can be used to pass instructions to the command interface. The buffer itself is located in a physical memory region. The ring buffer is defined by a 4 Dword register set that includes starting address, length, head offset, tail offset, and control information. Refer to the *Programming Interface* chapter for a detailed description of the parameters specified in this ring buffer register set, restrictions on the placement of ring buffer memory, arbitration rules, and in how the ring buffer can be used to pass instructions.

Ring Buffer Tail Offsets must be properly programmed before ring is enabled. A Ring Buffer can be enabled when empty.

Bit	Description
31:21	**Reserved** | Project: | All | Format: | MBZ
20:3	**Tail Offset** Project: | All Format: | U18 | QWord Offset This field is written by software to specify where the valid instructions placed in the ring buffer end. The value written points to the QWord *past* the last valid QWord of instructions. In other words, it can be defined as the *next* QWord that software will write instructions into. Software must write subsequent instructions to QWords following the Tail Offset, possibly wrapping around to the top of the buffer (i.e., software can't skip around within the buffer). Note that all DWords prior to the location indicated by the **Tail Offset** must contain valid instruction data – which may require instruction padding by software. See **Head Offset** for more information.
2:0	**Reserved** | Project: | All | Format: | MBZ

RING_BUFFER_HEAD

Register Type:	MMIO
Address Offset:	2034h
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

These registers are used to define and operate the "ring buffer" mechanism which can be used to pass instructions to the command interface. The buffer itself is located in a physical memory region. The ring buffer is defined by a 4 Dword register set that includes starting address, length, head offset, tail offset, and control information. Refer to the *Programming Interface* chapter for a detailed description of the parameters specified in this ring buffer register set, restrictions on the placement of ring buffer memory, arbitration rules, and in how the ring buffer can be used to pass instructions.

Ring Buffer Head Offsets must be properly programmed before ring is enabled. A Ring Buffer can be enabled when empty.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:21</td><td>
<table>
<tr><td colspan="3"><b>Wrap Count</b></td></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Default Value:</td><td>0h</td><td></td></tr>
<tr><td>Format:</td><td>U11</td><td>count of ring buffer wraps</td></tr>
<tr><td colspan="3">This field is incremented by 1 whenever the <b>Head Offset</b> wraps from the end of the buffer back to the start (i.e., whenever it wraps back to 0). Appending this field to the <b>Head Offset</b> field effectively creates a virtual 4GB Head "Pointer" which can be used as a tag associated with instructions placed in a ring buffer. The Wrap Count itself will wrap to 0 upon overflow.
The Wrap Count will get cleared as a result of writes of the Starting Address field.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th colspan="2">RING_BUFFER_HEAD</th></tr>
<tr><td>20:2</td><td>
<table>
<tr><th colspan="3">Head Offset</th></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Format:</td><td>U19</td><td>DWord Offset</td></tr>
<tr><td colspan="3">This field is written by software to specify where the valid instructions placed in the ring buffer end. The value written points to the QWord <i>past</i> the last valid QWord of instructions. In other words, it can be defined as the <i>next</i> QWord that software will write instructions into. Software must write subsequent instructions to QWords following the Tail Offset, possibly wrapping around to the top of the buffer (i.e., software can’t skip around within the buffer). Note that all DWords prior to the location indicated by the <b>Tail Offset</b> must contain valid instruction data – which may require instruction padding by software. See <b>Head Offset</b> for more information.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th><th>Project</th></tr>
<tr><td>A RB can be enabled empty or containing some number of valid instructions.</td><td>All</td></tr>
<tr><td>Head Offset is cleared as a result of writes of the Starting Address field.</td><td>All</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>0</td><td>
<table>
<tr><td><b>Wait for Condition Indicator</b></td><td>Project:</td><td>All</td><td>Format:</td><td>Enabled</td></tr>
<tr><td colspan="5">This is a read only value used to indicate whether or not the command streamer is currently waiting for a conditional code to be cleared from 0x2028</td></tr>
</table>
</td></tr>
</table>

RING_BUFFER_START

Register Type:	MMIO
Address Offset:	2038h
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

These registers are used to define and operate the “ring buffer” mechanism which can be used to pass instructions to the command interface. The buffer itself is located in a physical memory region. The ring buffer is defined by a 4 Dword register set that includes starting address, length, head offset, tail offset, and control information. Refer to the *Programming Interface* chapter for a detailed description of the parameters specified in this ring buffer register set, restrictions on the placement of ring buffer memory, arbitration rules, and in how the ring buffer can be used to pass instructions.

Bit	Description
31:12	**Starting Address** Project: All Address: GraphicsAddress[31:12] Surface Type: RingBuffer This field specifies Bits 31:12 of the 4KB-aligned starting Graphics Address of the ring buffer. Address bits 31 down to 29 must be zero. Writing this register also causes the Head Offset to be reset to zero, and the Wrap Count to be reset to zero. All ring buffer pages must map to Main Memory (uncached) pages. Ring Buffer addresses are always translated through the global GTT. Per-process address space can only be used via a batch buffer with the appropriate **Memory Space Select**.
11:0	**Reserved** Project: All Format: MBZ

RING_BUFFER_CONTROL

Register Type:	MMIO
Address Offset:	203Ch
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

These registers are used to define and operate the "ring buffer" mechanism which can be used to pass instructions to the command interface. The buffer itself is located in a physical memory region. The ring buffer is defined by a 4 Dword register set that includes starting address, length, head offset, tail offset, and control information. Refer to the *Programming Interface* chapter for a detailed description of the parameters specified in this ring buffer register set, restrictions on the placement of ring buffer memory, arbitration rules, and in how the ring buffer can be used to pass instructions.

Bit	Description
31:0	**Buffer Length** Project: All Format: U9-1 — Count of 4 KB pages Range 0..1FFh This field is written by SW to specify the length of the ring buffer in 4 KB Pages. Range = [0 = 1 page = 4 KB, 1FFh = 512 pages = 2 MB]
11	**RB Wait** — Project: All — Format: Boolean Indicates that this ring has executed a WAIT_FOR_EVENT instruction and is currently waiting. Software can write a "1" to clear this bit, write of "0" has no effect. When the RB is waiting for an event and this bit is cleared, the wait will be terminated and the RB will be returned to arbitration.
10:3	**Reserved** — Project: All — Format: MBZ

RING_BUFFER_CONTROL

2:1

Automatic Report Head Pointer	
Project:	All

This field is written by software to control the automatic “reporting” (write) of this ring buffer’s “Head Pointer” register (register DWord 1) to the corresponding location within the Hardware Status Page. Automatic reporting can either be disabled or enabled at 4KB, 64KB or 128KB boundaries within the ring buffer.

Value	**Name**	**Description**	**Project**
0h	MI_AUTOREPORT_OFF	Automatic reporting disabled	All
1h	MI_AUTOREPORT_64KB MI_AUTOREPORT_4KB	Report every 16 pages (64KB) When the **Per-Process Virtual Address Space and Context Queue Enable** bit is set, the ring buffer reports every 4KB	All
2h	Reserved	Reserved	All
3h	MI_AUTOREPORT_128KB	Report every 32 pages (128KB)	All

0

Ring Buffer Enable	Project:	All	Format:	Enable

This field is used to enable or disable this ring buffer. It can be enabled or disabled regardless of whether there are valid instructions pending.

8.5.1 UHPTR — Pending Head Pointer Register

UHPTR — Pending Head Pointer Register

Register Type:	MMIO
Address Offset:	2134h
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

Bit	Description
31:3	**Head Pointer Address** Project: All Default Value: 0h Address: GraphicsAddress[31:3] This register represents the GFX address offset where execution should continue in the ring buffer following execution of an MI_ARB_CHECK command.
2:1	**Reserved** Project: All Format: MBZ
0	**Head Pointer Valid** Project: All Default Value: 0h Format: U1 This bit is set by the software to request a pre-emption. It is reset by hardware after the head pointer in this register is read. The hardware uses the head pointer programmed in this register at the time the reset is generated.

Value	Name	Description	Project
0h		No valid updated head pointer register, resume execution at the current location in the ring buffer	All
1h		Indicates that there is an updated head pointer programmed in this register	All

8.6 Debug Registers Control

8.6.1 HW_MEMRD—Memory Read Sync Register (Debug)

HW_MEMRD—Memory Read Sync Register (Debug)

Register Type:	MMIO
Address Offset:	2060h
Project:	All
Default Value:	0000 0000h
Access:	RO
Size (in bits):	32

This register is used to flush the data from the Graphics dedicated chipset buffers into memory. A read to the register is generated post-flush completion of the graphics pipeline by the software. Read to this register is expected to be used in debug mode. The hardware will always return 0 for this register.

Bit	Description				
31:0	**Reserved**	Project:	All	Format:	MBZ

8.6.2 IPEIR—Instruction Parser Error Identification Register (Debug)

IPEIR—Instruction Parser Error Identification Register (Debug)

Register Type:	MMIO
Address Offset:	2064h
Project:	All
Default Value:	0000 0000h
Access:	RO
Size (in bits):	32

The IPEIR register identifies the general location of instructions that generate a Invalid Instruction Errors for the Renderer IP. (Note: The header (DWord 0) of the offending instruction will be stored in the IPEHR register).

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:4</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>3</td><td>
<table>
<tr><td><b>Batch Buffer Error</b></td><td>Project:</td><td>All</td><td>Format:</td><td>Flag</td></tr>
<tr><td colspan="5">If this bit is set the faulting instruction was executed from a batch buffer. If this bit is clear the faulting instruction was executed directly from a ring buffer.</td></tr>
</table>
</td></tr>
<tr><td>2:0</td><td>
<table>
<tr><td colspan="2"><b>Ring ID</b></td></tr>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>U3</td></tr>
<tr><td colspan="2">This field indicates which ring buffer is associated with the faulting</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0</td><td>Ring Buffer 0</td><td>Ring Buffer 0</td><td>All</td></tr>
<tr><td>1-7</td><td>Reserved</td><td>Reserved</td><td>All</td></tr>
</table>
</td></tr>
</table>

8.6.3 HW_MEMCWR—Memory Snoop Sync Register ([DevCTG])

HW_MEMCWR—Memory Snoop Sync Register

Register Type:	MMIO
Address Offset:	2064h
Project:	DevCTG+
Default Value:	0000 0000h
Access:	RO
Size (in bits):	32

This register is used to flush the data from the Graphics dedicated chipset buffers targeted for the cacheable space into memory. A write of any non-zero value to this register is generated in the Interrupt service routine to complete any pending cacheable writes to memory. The flush operation is considered completed when this register reads back a 0.

Bit	Description
31:0	**Sync Data** Project: DevCTG Format: U32 In order to flush any pending cacheable writes from the graphics an non-zero value needs to be written to this register. When a register read back returns a Zero, the flush operation is complete

8.6.4 IPEHR—Instruction Parser Error Header Register (Debug)

IPEHR—Instruction Parser Error Header Register (Debug)	
Register Type:	MMIO
Address Offset:	2068h
Project:	All
Default Value:	0000 0000h
Access:	RO
Size (in bits):	32

The IPEHR register is used to identify the instructions that generate Invalid Instruction Errors. This register is loaded with the header (DWord 0) of each instruction that is executed. It will therefore hold the header of an instruction that generates an Invalid Instruction Error.

Bit	Description
31:0	**Header** Project: All Format: U32 This field will contain the header (DWord 0) of a Renderer IP instruction that generates an Invalid Instruction Error.

8.6.5 INSTDONE—Instruction Stream Interface Done Register (Debug)

INSTDONE—Instruction Stream Interface Done Register (Debug)	
Register Type:	MMIO
Address Offset:	206Ch
Project:	All
Default Value:	FFE7 FFFEh
Access:	RO
Size (in bits):	32

This read-only register reports "Done" signals associated with the various internal engines and instruction transport mechanisms. In general, when the rendering engines of the device are idle, all bits will be set. If, for some reason, the device hangs, this register can be used to determine which functions are stalled with pending operations.

Bit	Description
31	**Row 0, EU 0 Done**
30	**Row 0, EU 1 Done**
29	**Row 0, EU 2 Done**
28	**Row 0, EU 3 Done**
27	**Row 1, EU 0 Done**
26	**Row 1, EU 1 Done**
25	**Row 1, EU 2 Done**
24	**Row 1, EU 3 Done**
23	**Strips and Fans (SF) Done**
22	**Setup (SE) Done**
21	**Windower (WM) Done**
20	**Reserved. Read as "0"**
19	**Reserved. Read as "0"**
18	**Dispatcher (DIP) Done**
17	**Projection and LOD (PL) Done**

INSTDONE—Instruction Stream Interface Done Register (Debug)

16	**Dependent Address Generator (DG) Done**
15	**Quad Cache Controller (QC) Done**
14	**Texture Fetch (FT) Done**
13	**Texture Decompressor (DM) Done**
12	**Sampler Cache (SC) Done**
11	**Filter (FL) Done**
10	**Bypass FIFO (BY) Done**
9	**Pixel Shader (PS) Done**
8	**Color Calculator (CC) Done**
7	**Map Filter Done: FL_done**
6	**Map L2 Cache Idle.**
5	**Message Arbiter Row 0 (EU output and EU input for Row 0) Done**
4	**Message Arbiter Row 1 (EU output and EU input for Row 1) Done**
3	**Instruction Cache Row 0 Done**
2	**Instruction Cache Row 1 Done**
1	**Command Parser (CP) Done**
0	**Ring 0 Enable**

8.6.6 INSTPS—Instruction Parser State Register (Debug)

INSTPS—Instruction Parser State Register (Debug)

Register Type:	MMIO
Address Offset:	2070h
Project:	All
Default Value:	UUUU UUUUh
Access:	RO
Size (in bits):	32

This register contains the state code of the Instruction Parser in the CSI. Decoding the contents of this register will indicate what the Instruction Parser is currently doing.

Bit	Description
31:0	**Instruction Parser State** | Project: All | Format: *Implementation Specific* Fields in this register identify the active Ring Buffer or Batch Buffer, and Batch buffer type.

8.6.7 ACTHD — Active Head Pointer Register (Debug)

ACTHD — Active Head Pointer Register (Debug)

Register Type:	MMIO
Address Offset:	2074h
Project:	All
Default Value:	0000 0000h
Access:	RO
Size (in bits):	32

This register contains the Head "Pointer" (DWord Graphics Memory Address) of the currently-active ring buffer.

Bit	Description
31:2	**Head Pointer** Project: All Default Value: 0h Address: GraphicsAddress[31:2] DWord Graphics Address corresponding to the Head Pointer of the currently-active ring or batch buffer.
1:0	**Reserved** Project: All Format: MBZ

8.6.8 DMA_FADD_P — Primary DMA Engine Fetch Address (Debug)

DMA_FADD_P — Primary DMA Engine Fetch Address (Debug)	
Register Type:	MMIO
Address Offset:	2078h
Project:	All
Default Value:	0000 0000h
Access:	RO
Size (in bits):	32

This register contains the QWord offset from the start address of the instruction being fetched by the Primary DMA engine.

Bit	Description
31:3	**Current DMA QWord Offset** | Project: | All | Format: | U30 This field contains the offset of the QWord (from the start of the ring buffer or batch buffer) that the “Primary” instruction parser DMA engine is currently accessing (fetching). Note that this offset will typically lead the Head offset (as instructions must be fetched before execution).
2:0	**Reserved** | Project: | All | Format: | MBZ

8.6.9 INSTDONE_1 — Additional Instruction Stream Interface Done (Debug)

INSTDONE_1 — Additional Instruction Stream Interface Done (Debug)	
Register Type:	MMIO
Address Offset:	207Ch
Project:	All
Default Value:	0000 0000h
Access:	RO
Size (in bits):	32

Bit	Description
31:20	**Reserved**
19	**gw_cs_done_cr**
18	**svsm_cs_done_cr**
17	**svdw_cs_done_cr**
16	**svdr_cs_done_cr**
15	**svrw_cs_done_cr**
14	**svrr_cs_done_cr**
13	**svts_cs_done_cr**
12	**masm_cs_done_cr**
11	**masf_cs_done_cr**
10	**mawb_cs_done_cr**
9	**em1_cs_done_cr**
8	**em0_cs_done_cr**
7	**uc1_cs_done**
6	**uc0_cs_done**
5	**urb_cs_done**
4	**isc_cs_done**

INSTDONE_1 — Additional Instruction Stream Interface Done (Debug)

3	**cl_cs_done**
2	**gs_cs_done**
1	**vs0_cs_done**
0	**vf_cs_done**

8.6.10 INSTDONE_1 — Additional Instruction Stream Interface Done (Debug)[DevCTG+]

INSTDONE_1 — Additional Instruction Stream Interface Done (Debug)	
Register Type:	MMIO
Address Offset:	207Ch
Project:	DevCTG+
Default Value:	0000 0000h
Access:	RO
Size (in bits):	32

Bit	Description
31	**BCS Done**
30	**CS Done**
29	**MASF Done**
28	**SVDW Done**
27	**SVDR Done**
26	**SVRW Done**
25	**SVRR Done**
24	**ISC Done**
23	**MT Done**
22	**RC Done**
21	**DAP Done**
20	**MAWB Done**
19	**MT Idle**
18	**GBLTbusy**
17	**SVSM Done**
16	**MASM Done**
15	**QC Done**
14	**FL Done**
13	**SC Done**
12	**DM Done**
11	**FT Done**
10	**DG Done**
9	**SI Done**

INSTDONE_1 — Additional Instruction Stream Interface Done (Debug)

8	**SO Done**
7	**PL Done**
6	**WIZ Done**
5	**URB Done**
4	**SF Done**
3	**CL Done**
2	**GS Done**
1	**VS0 Done**
0	**VF Done**

8.6.11 GFX_FLSH_CNTL — Graphics Flush Control

GFX_FLSH_CNTL — Graphics Flush Control

Register Type:	MMIO
Address Offset:	2170h
Project:	All
Default Value:	0000 0000 0000 0000h
Access:	Write Only
Size (in bits):	64

The flush initiated by this register is required whenever the GTT base address is changed or GTT entries are updated directly in memory by the host. See the description of the PGTBL_CTL_0 register for the sequence of operations required to update the GTT base or directly update GTT entries without using GTTADR.

Bit	Description
63:0	Project: All Format: U64 A CPU Dword/Qword write to this space flushes the GWB of all writes. The data associated with the write to this register is discarded. A command stream write to this space has no effect and the write data is discarded; the cycle is completed. It is UNDEFINED to read from this register.

8.6.12 CTXT_PREMP_DBG – Pre-emption Debug Register ([DevCTG] Only)

CTXT_PREMP_DBG – Pre-emption Debug Register	
Register Type:	MMIO
Address Offset:	2718h
Project:	DevCTG
Default Value:	0000 0000h
Access:	R/W (Debug Only)
Size (in bits):	32
This register is *not* saved or restored with context.	

Bit	Description
31:5	**Reserved** | Project: | DevCTG | Format: | MBZ
4	**Disable Marker Flow** | Project: | DevCTG | Format: | Disable If this bit is set, the flow of markers in the 3d pipeline is disabled. VF will not generate any VF Object End markers. WIZ will disable its super span counter. If clear, VF Object End markers are generated, propagated and processed as usual. If a new Run List is submitted when the **Disable Marker Flow** bit is set, the HW behavior should be functionally similar to the case when **Mid-Primitive Pre-Emption Disable** bit is set. Graphics reset value is zero. Unaffected by render pipeline reset. If this bit is modified, core behavior is undefined until the next render pipeline reset sequence.
3	**Debug Counter Replay Increment Disable** | Project: | DevCTG | Format: | Disable If this bit is set, the debug counters will not increment during replay of previously parsed commands due to a pre-emption. If clear, debug counters are always incremented whether or not the pipeline is in replay mode or not.
2	**Mid-Object Pre-emption Disable Chicken Bit** | Project: | DevCTG | Format: | Disable Setting this bit will prevent the submission of a new run list from pre-empting the current context in the middle of an object. If this bit is set, the Windower will complete all superspans for the current object (not just those that were past the commit point) prior to terminating execution of the current context and saving it. Note that replay will still be required for any remaining objects in the current primitive. This bit should normally be clear; setting it has the potential to increase pre-emption latency, especially for large triangles.
1	**Reserved** | Project: | All | Format: | MBZ
0	**Media Object Pre-emption Disable Chicken Bit** | Project: | DevCTG | Format: | Disable Setting this bit will prevent submission of a new run list from pre-empting the current context in the middle of a MEDIA_OBJECT command. Any such command that has started will complete before switching to the new run list. No replay will need to be done the next time the pre-empted context is re-submitted and switched to. This bit should normally be clear; setting it has the potential to greatly increase pre-emption latency.

8.7 NOPID — NOP Identification Register

NOPID — NOP Identification Register

Register Type:	MMIO
Address Offset:	2094h
Project:	All
Default Value:	00000000h
Access:	RO
Size (in bits):	32
Trusted Type:	1

The NOPID register contains the Noop Identification value specified by the last MI_NOOP instruction that enabled this register to be updated.

Bit	Description
31:22	**Reserved** | Project: | All | Format: | MBZ
21:0	**Identification Number** Project: All Security: None Default Value: 0h DefaultVaueDesc This field contains the 22-bit Noop Identification value specified by the last MI_NOOP instruction that enabled this field to be updated **Programming Notes** | **Project** This register is expected to be used for debug purposes to keep track of the execution of the command buffer | All

8.8 Watchdog Timer Registers [DevCTG]

These 2 registers together implement a watchdog timer. Writing ones to the control register enables the counter, and writing zeroes disables the counter. The 2nd register is programmed with a threshold value which, when reached, signals an interrupt then resets the counter to 0. Program the threshold value before enabling the counter or extremely frequent interrupts may result.

Note that the counter itself is not observable. It increments with the main render clock.

8.8.1 PR_CTR_CTL—Render Watchdog Counter Control

PR_CTR_CTL—Render Watchdog Counter Control

Register Type:	MMIO
Address Offset:	2178h
Project:	DevCTG
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

Bit	Description
31:0	**Counter logic op** | Project: | DevCTG | Format: | U32 This field specifies the action to be taken by the clock counter to generate interrupts. Writing 0 into this register causes a core render clock counter to be kicked off. Writing 1 into this register causes a core render clock counter to be stopped and reset to 0.

8.8.2 PR_CTR_THRSH—Render Watchdog Counter Threshold

PR_CTR_THRSH—Render Watchdog Counter Threshold

Register Type:	MMIO
Address Offset:	217Ch
Project:	DevCTG
Default Value:	0014 5855h
Access:	R/W
Size (in bits):	32

Bit	Description
31:0	**Counter logic Threshold** Project: DevCTG Format: U32 This field specifies the threshold that the hardware checks against for the value of the render clock counter before generating an interrupt. The counter in hardware generates an interrupt when the threshold is reached, rolls over and starts counting again. The interrupt generated is the “Media Hang Notify” interrupt since this watchdog timer is intended primarily to remedy VLD hangs on the main pipeline.

8.8.3 PR_CTR—Render Watchdog Counter

PR_CTR—Render Watchdog Counter

Register Type:	MMIO
Address Offset:	2190h
Project:	DevCTG
Default Value:	0000 0000h
Access:	RO
Size (in bits):	32

Bit	Description
31:0	**Counter Value** Project: DevCTG Format: U32 This register reflects the render watchdog counter value itself. It cannot be written to but can be read for debug purposes.

8.9 Interrupt Control Registers

The Interrupt Control Registers described below all share the same bit definition. The bit definition is as follows:

Table 8-1. Bit Definition for Interrupt Control Registers

Bit	Description
31:26	**Reserved.** These bits may be assigned to interrupts on future products/steppings.
25	**Video Decode Command Parser User Interrupt:** This status bit is set when an MI_USER_INTERRUPT instruction is executed on the Media Decode Command Parser. Note that instruction execution is not halted and proceeds normally. A mechanism such as an MI_STORE_DATA instruction is required to associate a particular meaning to a user interrupt. [DevBW] and [DevCL]: This bit is reserved.
24	**Context Switch Interrupt:** Set when a context switch has just occurred. **Per-Process Virtual Address Space and Run List Enable bit** needs to be set for this interrupt to occur**.** [DevBW], [DevCL]: This bit is reserved.
23	**Page Fault:** This bit is set whenever there is a pending PPGTT (page or directory) fault. [DevBW], [DevCL]: This bit is reserved.
22	**Display A VLine Interrupt**: Set when programmed scan line is reached during Display fetch. [DevBW], [DevCL]: This bit is reserved.
21	**Display B VLine Interrupt**: Set when programmed scan line is reached during Display fetch. [DevBW], [DevCL]: This bit is reserved.
20	**Media Decode Pipeline Counter Exceeded Notify Interrupt:** The counter threshold for the execution of the media pipeline is exceeded. Driver needs to attempt hang recovery. [DevBW] and [DevCL]: This bit is reserved.
19	**Bit Stream Pipeline Counter Exceeded Notify Interrupt:** The counter threshold for the execution of the Bit Stream Pipeline is exceeded. Driver needs to attempt hang recovery. [DevBW] and [DevCL]: This bit is reserved.
18	**PIPE_CONTROL Notify Interrupt:** The Pipe Control packet (Fences) specified in *3D pipeline* document may optionally generate an Interrupt. The Store QW associated with a fence is completed ahead of the MSI. This ordering is not guaranteed if PCI Line Intr# mechanism is used.
17	Reserved
16	**Reserved.** MBZ

15	**Render Command Parser Master Error:** When this status bit is set, it indicates that the hardware has detected an error. It is set by the device upon an error condition and cleared by a CPU write of a one to the appropriate bit contained in the Error ID register followed by a write of a one to this bit in the IIR. Further information on the source of the error comes from the "Error Status Register" which along with the "Error Mask Register" determine which error conditions will cause the error status bit to be set and the interrupt to occur. **Page Table Error:** Indicates a page table error. **Instruction Parser Error**: The Renderer Instruction Parser encounters an error while parsing an instruction.
14	**GMCH Thermal Sensor Event:** This bit is set on "thermal events" detected by the Thermal Sensor logic.
13	**Reserved.** MBZ
12	**Sync Status:** This bit is toggled when the Instruction Parser completes a flush with the sync enable bit active in the INSTPM register. The toggle event will happen after all the graphics engines are flushed. The HW Status DWord write resulting from this toggle will cause the CPU's view of graphics memory to be coherent as well (flush and invalidate the render cache).
11	**Display Plane A Flip Pending:** This status bit is set on a Display Plane A pending flip (i.e., resulting from the execution of an MI_DISPLAY_BUFFER_INFO instruction). This is only used when the MI_DISPLAY_BUFFER _INFO instruction is being used. See that instruction for additional information.
10	**Display Plane B Flip Pending:** Flip Pending status for Display B. See **Display Plane A Flip Pending**
9	**Overlay Plane Flip Pending:** This status bit is set to reflect a pending overlay plane flip (i.e., resulting from the execution of an MI_OVERLAY_FLIP instruction). This is only affected by the use of MI_OVERLAY_FLIP instructions and not through the manual method.
8	**Display Plane C Flip Pending:** Flip Pending status for Display Plane C. See **Display Plane A Flip Pending**
7	**Display Pipe A VBLANK:** This status bit is set at leading edge of Display Pipe A VBLANK, though delayed to allow all internal hardware VBLANK events to occur before the interrupt is generated (to avoid race conditions). These events include the update of the display and overlay status bits and loading of the overlay registers. **[DevCL] If trunk clock gating is enabled, this interrupt should never be used.**
6	**Display Pipe A Event:** This status bit is set by the device on the active-going edge of the OR of unmasked Display Pipe A event bits. The specific cause of the event can be determined by reading the display status register. Note that the display line compare status can also be observed through the instruction interface.
5	**Display Pipe B VBLANK:** This status bit is set at leading edge of Display B VBLANK. This is actually delayed to allow all VBLANK events to occur before the interrupt is generated. These events include the update of the overlay registers. **[DevCL] If trunk clock gating is enabled, this interrupt should never be used.**

4	**Display Pipe B Event:** This status bit is set by the device on the active-going edge of the OR of unmasked Display Pipe A event bits. The specific cause of the event can be determined by reading the display status register. Note that the display line compare status can also be observed through the instruction interface.
3	**Reserved.** MBZ
2	**Debug Interrupt:** When this bit is set, the EU is indicating that it has encountered an interrupt in the kernel program. Refer to the GenX Debug PRM for more details
1	**Render Command Parser User Interrupt:** This status bit is set when an MI_USER_INTERRUPT instruction is executed on the Render Command Parser. Note that instruction execution is not halted and proceeds normally. A mechanism such as an MI_STORE_DATA instruction is required to associate a particular meaning to a user interrupt.
0	**ASLE Interrupt:** This status bit is set when ASLE (PCI Configuration register Device 2, Function 0, E4) is written by the System BIOS (any byte or all). The meaning of the interrupt is determined by the contents written.

The following table specifies the settings of interrupt bits stored upon a "Hardware Status Write" due to ISR changes.

Bit	Interrupt Bit	ISR bit Reporting via Hardware Status Write (when unmasked via HWSTAM)
25	**Video Decode Command Parser User Interrupt:** This status bit is set when an MI_USER_INTERRUPT instruction is executed on the Media Decode Command Parser. Note that instruction execution is not halted and proceeds normally. A mechanism such as an MI_STORE_DATA instruction is required to associate a particular meaning to a user interrupt. **[DevBW] and [DevCL]: This bit is reserved.**	Not supported to be unmasked
24	**Context Switch Interrupt:** Set when a context switch has just occurred. **[DevBW], [DevCL]: This bit is reserved.**	Not supported to be unmasked
23	**Page Fault:** This bit is set whenever there is a pending PPGTT (page or directory) fault. **[DevBW], [DevCL]: This bit is reserved.**	Set when event occurs, cleared when event cleared
22	**Display A VLine Interrupt**: Set when programmed scan line is reached during Display fetch. **[DevBW], [DevCL]: This bit is reserved.**	Not supported to be unmasked
21	**Display B VLine Interrupt**: Set when programmed scan line is reached during Display fetch. **[DevBW], [DevCL]: This bit is reserved.**	Set when event occurs, cleared when event cleared
20	**Media Decode Pipeline Counter Exceeded Notify Interrupt:** The counter threshold for the execution of the media pipeline is exceeded. Driver needs to attempt hang recovery. **[DevBW] and [DevCL]: This bit is reserved.**	Not supported to be unmasked

19	**Bit Stream Pipeline Counter Exceeded Notify Interrupt:** The counter threshold for the execution of the Bit Stream Pipeline is exceeded. Driver needs to attempt hang recovery. **[DevBW] and [DevCL]: This bit is reserved.**	Not supported to be unmasked
18	PIPE_CONTROL packet - Notify Enable	0
17	Reserved. MBZ	Set when event occurs, cleared when event cleared
16	Reserved. MBZ	0
15	Master Error	Set when error occurs, cleared when error cleared
14	GMCH Thermal Sensor Event	Should always be disabled for Hardware Status Write reporting.
13	Reserved. MBZ	0
12	Sync Status	Toggled every SyncFlush Event
11	Display Plane A Flip Pending	Set when flip is pending
10	Display Plane B Flip Pending	Set when flip is pending
9	[DevBW] and [DevCL] Only: Overlay Flip Pending [DevCTG] Only: Display Sprite B Flip Pending	Set when flip is pending
8	[DevBW] and [DevCL] Only: Display Plane C Flip Pending [DevCTG] Only: Display Sprite A Flip Pending	Set when Flip requested, cleared when flip occurs.
7	Display Pipe A VBlank	0
6	Display Pipe A Event	Set when event occurs, cleared when event cleared
5	Display Pipe B VBlank	0
4	Display Pipe B Event	Set when event occurs, cleared when event cleared
3	Reserved. MBZ	0
2	Debug Interrupt	Set when debug interrupt occurs.
1	User Interrupt	0
0	ASLE Interrupt	0

8.9.1 HWS_PGA — Hardware Status Page Address Register

HWS_PGA — Hardware Status Page Address Register

Register Type:	MMIO
Address Offset:	2080h
Project:	All
Default Value:	1FFFF000h
Access:	R/W
Size (in bits):	32
Trusted Type:	1

This register is used to program the 4 KB-aligned System Memory address of the Hardware Status Page used to report hardware status into (typically cacheable) System Memory.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:12</td><td>
<table>
<tr><th colspan="2">Address</th></tr>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Security:</td><td>None</td></tr>
<tr><td>Address:</td><td>PhysicalAddress[31:12]</td></tr>
<tr><td>Surface Type:</td><td>U32</td></tr>
<tr><td>Range</td><td>0..2^32-1</td></tr>
<tr><td colspan="2">This field is used by SW to specify Bits 31:12 of the 4 KB-aligned System Memory address of the 4 KB page known as the “Hardware Status Page”. The system address space is expected to be cacheable in memory.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Notes:
If the <b>Per-Process Virtual Address Space and Run List Enable</b> is set, HW requires that the status page is programmed to allow for the context switch status to be reported</td></tr>
</table>
</td></tr>
<tr><td>12:8</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>7:4</td><td>
<table>
<tr><th colspan="2">Physical Start Address Extension</th></tr>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Security:</td><td>None</td></tr>
<tr><td>Address:</td><td>PhysicalAddress[35:32]</td></tr>
<tr><td colspan="2">This field specifies Bits 35:32 of the starting physical address.</td></tr>
</table>
</td></tr>
<tr><td>3:0</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>3:1</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>DevCTG, HVN/ABD</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
</table>

HWS_PGA — Hardware Status Page Address Register

<table>
<tr><td>0</td><td>
<table>
<tr><td colspan="3"><b>Translation In Progress</b></td></tr>
<tr><td>Project:</td><td colspan="2">DevCTG, HVN/ABD</td></tr>
<tr><td>Format:</td><td>U1</td><td>FormatDesc</td></tr>
<tr><td colspan="3">This field indicates that the translation for the hardware status page from the graphics virtual address to the physical address is pending. Software can use this indicator to prevent updating the status page when there is a pending cycle for translation.</td></tr>
</table>
</td></tr>
</table>

The following table defines the layout of the Hardware Status Page:

DWord Offset	Description
0	**Interrupt Status Register Storage:** The content of the ISR register is written to this location whenever an "unmasked" bit of the ISR (as determined by the HWSTAM register) changes state.
3:1	**Reserved.** Must not be used.
4	**Ring Head Pointer Storage:** The contents of the Ring Buffer Head Pointer register (register DWord 1) are written to this location either as result of an MI_REPORT_HEAD instruction or as the result of an "automatic report" (see RINGBUF registers).
Fh:5h	**Reserved.** Must not be used.
10h-1Bh	**Context Status DWords [DevCTG] Only.**
1Ch-1Eh	**Reserved.** Must not be used.
1Fh	**Reserved.** Must not be used.
20h-3FFh	These locations can be used for general purpose via the MI_STORE_DATA_INDEX or MI_STORE_DATA_IMM instructions.

8.9.2 PWRCTXA — Power Context Register Address ([DevCL] Only)

PWRCTXA — Power Context Register Address

Register Type:	MMIO
Address Offset:	2088h
Project:	DevCL
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

The PWRCTXA register has the address of the Global GTT translated memory location which stores the hardware context if the voltage is removed from the render clock well. The format of the hardware “power” context is specified in the Memory Data Formats.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:12</td><td>
<table>
<tr><th colspan="2">Power Context Address</th></tr>
<tr><td>Project:</td><td>DevCL</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Address:</td><td>GraphicsAddress[31:12]</td></tr>
<tr><td colspan="2">This field is used by SW to specify Bits 31:12 of the 4 KB-aligned Graphics Memory address. The graphics memory address is translated using the Global GTT.</td></tr>
</table>
</td></tr>
<tr><td>11:5</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>DevCL</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>4:1</td><td>
<table>
<tr><th colspan="2">Power Context Size</th></tr>
<tr><td>Project:</td><td>DevCL</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>U4</td></tr>
<tr><td colspan="2">Field specifies the size of the power context allocated by the software. The size is in terms of 4K pages
This field is ReadOnly</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>001-111</td><td>Reserved</td><td>Reserved</td><td>All</td></tr>
<tr><td>000</td><td>4KB</td><td></td><td>DevCL</td></tr>
</table>
</td></tr>
</table>

PWRCTXA — Power Context Register Address

0	**Power Context Enable**

Project:	DevCL
Default Value:	0h
Format:	Enable

This field determines whether the power context is enabled. If enabled, the Power Context Address specifies the starting address of the hardware context in memory. If the power context is not enabled, the hardware will disable reducing the render voltage.

Value	Name	Description	Project
0h	Disable	DISABLED	DevCL
1h	Enable	ENABLED	DevCL

8.9.3 HWSTAM — Hardware Status Mask Register

Hardware Status Mask Register	
Register Type:	MMIO
Address Offset:	2098h
Project:	All
Default Value:	**FFFE DFFFh**
Access:	R/W
Size (in bits):	32
Trusted Type:	1

The HWSTAM register has the same format as the Interrupt Control Registers. The bits in this register are "mask" bits that prevent the corresponding bits in the Interrupt Status Register from generating a "Hardware Status Write" (PCI write cycle). Any unmasked interrupt bit (HWSTAM bit set to 0) will allow the Interrupt Status Register to be written to the ISR location (within the memory page specified by the Hardware Status Page Address Register) when that Interrupt Status Register bit changes state.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:13</td><td>
<table><tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MB1</td></tr></table>
</td></tr>
<tr><td>12</td><td>
<table>
<tr><td colspan="3"><b>Sync Status</b></td></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Security:</td><td colspan="2">None</td></tr>
<tr><td>Default Value:</td><td>1h</td><td>Masked by default</td></tr>
<tr><td colspan="3">When this mask bit is clear, a change in the corresponding ISR bit will trigger a DWord write of the ISR contents to the "ISR location" in the Hardware Status Page. When a bit in this mask is set, a write will not be triggered by a change in the corresponding ISR bit.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th><th>Project</th></tr>
<tr><td>This bit is toggled when the Instruction Parser completes a flush with the sync enable bit active in the INSTPM register. The toggle event will happen after all the graphics engines are flushed. The HW Status DWord write resulting from this toggle will cause the CPU's view of graphics memory to be coherent as well (flush and invalidate the render cache).</td><td>All</td></tr>
</table>
</td></tr>
</table>

Hardware Status Mask Register

11

Display Plane A Flip Pending		
Project:	All	
Security:	None	
Default Value:	1h	Masked by default
When this mask bit is clear, a change in the corresponding ISR bit will trigger a DWord write of the ISR contents to the "ISR location" in the Hardware Status Page. When a bit in this mask is set, a write will not be triggered by a change in the corresponding ISR bit.		

Programming Notes	Project
This status bit is set on a Display Plane A pending flip (i.e., resulting from the execution of an MI_DISPLAY_BUFFER_INFO instruction). This is only used when the MI_DISPLAY_BUFFER _INFO instruction is being used. See that instruction for additional information.	All

10

Display Plane B Flip Pending		
Project:	All	
Security:	None	
Default Value:	1h	Masked by default
When this mask bit is clear, a change in the corresponding ISR bit will trigger a DWord write of the ISR contents to the "ISR location" in the Hardware Status Page. When a bit in this mask is set, a write will not be triggered by a change in the corresponding ISR bit.		

Programming Notes	Project
This status bit is set on a Display Plane B pending flip (i.e., resulting from the execution of an MI_DISPLAY_BUFFER_INFO instruction). This is only used when the MI_DISPLAY_BUFFER _INFO instruction is being used. See that instruction for additional information.	All

9

Overlay Plane Flip Pending		
Project:	All	
Security:	None	
Default Value:	1h	Masked by default
When this mask bit is clear, a change in the corresponding ISR bit will trigger a DWord write of the ISR contents to the "ISR location" in the Hardware Status Page. When a bit in this mask is set, a write will not be triggered by a change in the corresponding ISR bit.		

Programming Notes	Project
This status bit is set to reflect a pending overlay plane flip (i.e., resulting from the execution of an MI_OVERLAY_FLIP instruction). This is only affected by the use of MI_OVERLAY_FLIP instructions and not through the manual method.	All

Hardware Status Mask Register

8

Display Plane C Flip Pending		
Project:	All	
Security:	None	
Default Value:	1h	Masked by default
When this mask bit is clear, a change in the corresponding ISR bit will trigger a DWord write of the ISR contents to the "ISR location" in the Hardware Status Page. When a bit in this mask is set, a write will not be triggered by a change in the corresponding ISR bit.		

Programming Notes	Project
Flip Pending status for Display Plane C. See **Display Plane A Flip Pending**	All

8

Display Sprite A Flip Pending		
Project:	**DevCTG only**	
Security:	None	
Default Value:	1h	Masked by default
When this mask bit is clear, a change in the corresponding ISR bit will trigger a DWord write of the ISR contents to the "ISR location" in the Hardware Status Page. When a bit in this mask is set, a write will not be triggered by a change in the corresponding ISR bit.		

Programming Notes	Project
Flip Pending status for Display Sprite A. See **Display Plane A Flip Pending**.	**DevCTG +**

7:0

Reserved	Project:	All	Format:	MB1

8.9.4 IER — Interrupt Enable Register

IER — Interrupt Enable Register

Register Type:	MMIO
Address Offset:	20A0h
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

The IER register contains an interrupt enable bit for each interrupt bit in the IIR register. A disabled interrupt will still appear in the Interrupt Identity Register to allow polling of interrupt sources.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:0</td><td>
<table>
<tr><th colspan="3">Interrupt Enable Bits</th></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Default Value:</td><td colspan="2">0h</td></tr>
<tr><td>Format:</td><td>Array of Enables</td><td>Refer to the Interrupt Control Register section for bit definitions</td></tr>
<tr><td colspan="3">The bits in this register enable a CPU interrupt to be generated whenever the corresponding bit in the Interrupt Identity Register becomes set.</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>Disable</td><td>DISABLED</td><td>All</td></tr>
<tr><td>1h</td><td>Enable</td><td>ENABLED</td><td>All</td></tr>
</table>
</td></tr>
</table>

8.9.5 IIR — Interrupt Identity Register

IIR — Interrupt Identity Register

Register Type:	MMIO
Address Offset:	20A4h
Project:	All
Default Value:	0000 0000h
Access:	R/WC
Size (in bits):	32

The IIR register contains the interrupt bits that are “unmasked” by the IMR and thus can generate CPU bits (if enabled via the IER). When a CPU interrupt is generated, this should be the first register to be interrogated to determine the source of the interrupt. **Writing a ‘1’ into the appropriate bit position within this register clears interrupts**.

Programming Note: Prior to clearing a Display Pipe-sourced interrupt (e.g., Display Pipe A VBLANK) in the IIR, the corresponding interrupt (source) status in the PIPEASTAT register (e.g., Pipe A VBLANK Interrupt Status bit of PIPEASTAT) must first be cleared. Note that clearing these status bits requires writing a ‘1’ to the appropriate bit position.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:0</td><td>
<table>
<tr><th colspan="3">Interrupt Identity Bits</th></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Default Value:</td><td colspan="2">0h</td></tr>
<tr><td>Format:</td><td>Array of unmasked</td><td>Persistent interrupt bits</td></tr>
<tr><td colspan="3">This field holds the persistent values of the interrupt bits from the ISR which are “unmasked” by the IMR. If enabled by the IER, bits set in this register will generate a CPU interrupt. Bits set in this register will remain set (persist) until the interrupt condition is “cleared” via software by writing a ‘1’ to the appropriate bit(s).</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>1h</td><td>Interrupt Condition Detected</td><td>Interrupt Condition Detected (may or may not have actually generated a CPU interrupt)</td><td>All</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Bit 12 of the Interrupt Identity register is used for the sync status flush. The hardware toggles the bit at the completion of the flush. It is not expected that this bit will be used to generate interrupt. In case an interrupt is desired, software needs to toggle the bit back to 0 (by programming another sync flush) before clearing the IIR.</td></tr>
</table>
</td></tr>
</table>

8.9.6 IMR—Interrupt Mask Register

IMR—Interrupt Mask Register	
Register Type:	MMIO
Address Offset:	20A8h
Project:	All
Default Value:	FFFE DFFFh
Access:	R/W
Size (in bits):	32

The IMR register is used by software to control which Interrupt Status Register bits are “masked” or “unmasked”. “Unmasked” bits will be reported in the IIR, possibly triggering a CPU interrupt, and will persist in the IIR until cleared by software. “Masked” bits will not be reported in the IIR and therefore cannot generate CPU interrupts.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>The live bspec is posted on MOSS31:0</td><td>
<table>
<tr><th colspan="3">Interrupt Mask Bits</th></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Default Value:</td><td colspan="2">FFFE DFFFh</td></tr>
<tr><td>Format:</td><td>Array of interrupt mask bits</td><td>Refer to Table 8-1 in Interrupt Control Register section for bit definitions</td></tr>
<tr><td colspan="3">This field contains a bit mask which selects which interrupt bits (from the ISR) are reported in the IIR.</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>Not Masked</td><td>Will be reported in the IIR</td><td>All</td></tr>
<tr><td>1h</td><td>Masked</td><td>Will not be reported in the IIR</td><td>All</td></tr>
</table>
</td></tr>
</table>

8.9.7 ISR — Interrupt Status Register

ISR — Interrupt Status Register

Register Type:	MMIO
Address Offset:	20ACh
Project:	All
Default Value:	0000 0000h
Access:	RO
Size (in bits):	32

The ISR register contains the non-persistent value of all interrupt status bits. The IMR register selects which of these interrupt conditions are reported in the persistent IIR (i.e., set bits must be cleared by software). Bits in the IER are used to selectively enable IIR bits to cause CPU interrupts.

Programming Note: The User Interrupt bit in this register is a short pulse therefore software should not expect to use this register to sample these conditions.

Bit	Description
31:0	**Interrupt Status Bits**

Project:	All	
Default Value:	0h	
Format:	Array of interrupt status bits	Refer to Table 8-1 in Interrupt Control Register section for bit definitions

This field contains the non-persistent values of all interrupt status bits.

Value	Name	Description	Project
1h	Interrupt Condition Exists	Interrupt Condition currently exists	All

8.10 Hardware-Detected Error Bit Definitions (for EIR, EMR, ESR)

This section defines the Hardware-Detected Error bit definitions and ordering that is common to the EIR, EMR and ESR registers. The EMR selects which error conditions (bits) in the ESR are reported in the EIR. Any bit set in the EIR will cause the Master Error bit in the ISR to be set. EIR bits will remain set until the appropriate bit(s) in the EIR is cleared by writing the appropriate EIR bits with '1'.

The following table describes the Hardware-Detected Error bits:

Table 8-2. Hardware-Detected Error Bits

Bit	Description
15:10	**[DevCTG] BCS Page Table Error**: This bit is set when a Graphics Memory Mapping Error is detected on the Video Decode Command Parser. The cause of the error is indicated (to some extent) in the PGTBL_ER register. Note: This error indication can not be cleared except by reset (i.e., it is a fatal error). 1 = Page table error [DevBW], [DevCL]: Reserved.
9	**[DevCTG] BCS Instruction Error:** This bit is set when the Media Decode Command Parser detects an error while parsing a command. Instruction errors include: 1) Client ID value (Bits 31:29 of the Header) is not supported. 2) Undefined Command Opcodes: The (*debug*) INSTPS register may provide more information as to the cause of the error. The (*debug*) IPEHR register contains the header (DWord 0) of the faulting instruction. The (*debug*) BCS_IPEIR, and BCS_DMA_FADD registers provide an indication of where the faulting instruction is located and which instruction stream mechanism caused the instruction to be executed. 1: Instruction Error detected [DevBW], [DevCL]: Reserved.
8	Reserved: MBZ
7:6	**AVC Error Detected:** When this bit is set, it indicates the AVC Video Decode Fixed Function has detected an error. Further information on the source of this error comes from the "AVC Error Status Register" which determines which error condition caused the AVC error status bit to be set and the interrupt to occur. [DevBW] and [DevCL]: This bit is reserved.
5	**Page Table Error**: This bit is set when a Graphics Memory Mapping Error is detected. The cause of the error is indicated (to some extent) in the PGTBL_ER register. Note: This error indications can not be cleared except by reset (i.e., it is a fatal error). 1 = Page table error

Bit	Description
4	**Memory Privilege Violation Error [DevCTG] Only.** This bit is set if a command in a non-secure batch buffer attempts an operation to the GGTT (this can only happen in commands that contain a PPGTT vs. GGTT selector). The command will be executed as if the selector bit indicated PPGTT and parsing will continue. [DevBW], [DevCL]: Reserved.
3	**Command Privilege Violation Error [DevCTG] Only.** This bit is set if a command classified as privileged is parsed in a non-secure batch buffer. The command will be converted to a NOOP and parsing will continue. [DevBW], [DevCL]: Reserved.
2	**[DevCTG] BCS Page Table Error**: This bit is set when a Graphics Memory Mapping Error is detected on the Video Decode Command Parser. The cause of the error is indicated (to some extent) in the PGTBL_ER register. Note: This error indication can not be cleared except by reset (i.e., it is a fatal error). 1 = Page table error [DevBW], [DevCL]: Reserved.
1	**Main Memory Refresh Timer Error:** This bit is set when the device detects a timeout related to refreshing Main Memory. [DevBW]: Reserved.
0	**Instruction Error:** This bit is set when the Renderer Instruction Parser detects an error while parsing an instruction. Instruction errors include: 1) Client ID value (Bits 31:29 of the Header) is not supported (only MI, 2D and 3D are supported). 2) Defeatured MI Instruction Opcodes: The (*debug*) INSTPS register may provide more information as to the cause of the error. The (*debug*) IPEHR register contains the header (DWord 0) of the faulting instruction. The (*debug*) IPEIR, BBP_PTR, ABB_PTR, ABB_END and DMA_FADD registers provide an indication of where the faulting instruction is located and which instruction stream mechanism caused the instruction to be executed. 1: Instruction Error detected **Programming Note:** The bit for the error mask of this register is reserved. The mask should be set to a value of 1.

8.10.1 EIR — Error Identity Register

<table>
<tr><th colspan="2">EIR — Error Identity Register</th></tr>
<tr><td><b>Register Type:</b></td><td>MMIO</td></tr>
<tr><td><b>Address Offset:</b></td><td>20B0h</td></tr>
<tr><td><b>Project:</b></td><td>All</td></tr>
<tr><td><b>Default Value:</b></td><td>0000 0000h</td></tr>
<tr><td><b>Access:</b></td><td>R/WC</td></tr>
<tr><td><b>Size (in bits):</b></td><td>32</td></tr>
<tr><td colspan="2">The EIR register contains the persistent values of Hardware-Detected Error Condition bits. Any bit set in this register will cause the Master Error bit in the ISR to be set. The EIR register is also used by software to clear detected errors (by writing a ‘1’ to the appropriate bit(s)).</td></tr>
</table>

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:16</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>15:0</td><td>
<table>
<tr><th colspan="3">Error Identity Bits</th></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Default Value:</td><td colspan="2">0h</td></tr>
<tr><td>Format:</td><td>Array of Error condition bits</td><td>See Table 8-2 Hardware-Detected Error Bits</td></tr>
<tr><td colspan="3">This register contains the persistent values of ESR error status bits that are unmasked via the EMR register. The logical OR of all (defined) bits in this register is reported in the Master Error bit of the Interrupt Status Register. In order to clear an error condition, software must first clear the error by writing a ‘1’ to the appropriate bit(s) in this field. If required, software should then proceed to clear the Master Error bit of the IIR.</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>1h</td><td>Error occurred</td><td>Error occurred</td><td>All</td></tr>
</table>
<table>
<tr><th>Programming Notes</th><th>Project</th></tr>
<tr><td>Writing a ‘1’ to a set bit will cause that error condition to be cleared. However, the Page Table Error bit (Bit 4) cannot be cleared except by reset (i.e., it is a fatal error).</td><td>All</td></tr>
</table>
</td></tr>
</table>

8.10.2 EMR—Error Mask Register

EMR—Error Mask Register

Register Type:	MMIO
Address Offset:	20B4h
Project:	All
Default Value:	FFFF FFDFh
Access:	R/W
Size (in bits):	32

The EMR register is used by software to control which Error Status Register bits are "masked" or "unmasked". "Unmasked" bits will be reported in the EIR, thus setting the Master Error ISR bit and possibly triggering a CPU interrupt, and will persist in the EIR until cleared by software. "Masked" bits will not be reported in the EIR and therefore cannot generate Master Error conditions or CPU interrupts.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:16</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>15:0</td><td>
<table>
<tr><td colspan="3"><b>Error Mask Bits</b></td></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Default Value:</td><td colspan="2">FFFF FFDFh</td></tr>
<tr><td>Format:</td><td>Array of error condition mask bits</td><td>See Table 8-2. Hardware-Detected Error Bits</td></tr>
<tr><td colspan="3">This register contains a bit mask that selects which error condition bits (from the ESR) are reported in the EIR.</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>Not Masked</td><td>Will be reported in the EIR</td><td>All</td></tr>
<tr><td>1h</td><td>Masked</td><td>Will not be reported in the EIR</td><td>All</td></tr>
</table>
</td></tr>
</table>

8.10.3 ESR—Error Status Register

ESR—Error Status Register	
Register Type:	MMIO
Address Offset:	20B8h
Project:	All
Default Value:	0000 0000h
Access:	RO
Size (in bits):	32

The ESR register contains the current values of all Hardware-Detected Error condition bits (these are all by definition "persistent"). The EMR register selects which of these error conditions are reported in the persistent EIR (i.e., set bits must be cleared by software) and thereby causing a Master Error interrupt condition to be reported in the ISR.

Bit	Description				
31:16	**Reserved**	Project:	All	Format:	MBZ
15:0	**Error Status Bits**				

Error Status Bits		
Project:	All	
Default Value:	0h	
Format:	Array of error condition bits	See Table 8-2. Hardware-Detected Error Bits

This register contains the non-persistent values of all hardware-detected error condition bits.

Value	Name	Description	Project
1h	Error Condition Detected	Error Condition detected	All

8.11 Probe List Registers ([DevCTG] Only)

Surface probing is a procedure performed at the beginning of a rendering sequence (command buffer) to verify that all required surfaces in a process' virtual address space are actually present in physical memory prior to beginning the sequence. A different process can then be switched to and run while the required surfaces are being brought into memory (by SW). The registers here work in concert with the probe commands (see Memory Interface Commands for Rendering) to provide this interface. "Slots" are the designated places in a processes' context image where probes (surface base addresses) are stored. The stored probes are used by SW to determine which surfaces a context requires, and are also used by HW to re-validate that surfaces are resident upon a context restore.

See MI_PROBE in *Memory Interface Commands for Rendering* for more details.

8.11.1 PRBL_SF – Probe List Slot Fault Register

PRBL_SF – Probe List Slot Fault Register

Register Type:	MMIO
Address Offset:	2680h
Project:	DevCTG
Default Value:	0000 0000h
Access:	RO
Size (in bits):	64

This register contains the fault bits for the probe slots, one bit for each cacheline of the 1024 probe slot memory area. It cannot be directly written by SW. The image of this register in the per-process HW status page can be read after a context switch (due to surface fault) to determine which cachelines of the probe list contain faulting probes. This register is saved with context. It is not restored but recomputed while re-validating the probe list on a context restore.

Bit	Description				
63:0	**Slot Fault Line 63:0**	Project:	DevCTG	Format:	Array:Enable
	If set, indicates that the corresponding probe list cacheline (in memory) contains a probe that has faulted.				

8.12 Register Definitions for Context Save

8.12.1 INSTPM—Instruction Parser Mode Register

INSTPM—Instruction Parser Mode Register

Register Type:	MMIO
Address Offset:	20C0h
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Trusted Type:	1

The INSTPM register is used to control the operation of the Instruction Parser. Certain classes of instructions can be disabled (ignored) – often useful for detecting performance bottlenecks. Also, “Synchronizing Flush” operations can be initiated – useful for ensuring the completion (vs. only parsing) of rendering instructions.

Programming Notes:

- If an instruction type is disabled, the parser will read those instructions but not process them.
- Error checking will be performed even if the instruction is ignored.
- All Reserved bits are implemented.
- This Register is saved and restored as part of Context.

Bit	Description
31:16	**Mask Bits** Format: Mask[15:0] **Masks:** These bits serve as write enables for bits 15:0. If this register is written with any of these bits clear the corresponding bit in the field 15:0 will not be modified. Reading these bits always returns 0s.
15:11	**Reserved** Project: All Format: MBZ
10	**Forward Progress Disable** Project: DevCTG+ Security: None Default Value: 0h DefaultVaueDesc Mask: MMIO(0x20C0)#26 When this bit is set, the hardware will allow a pre-empt interrupt to happen even when there is no forward progress on the current context. Only valid when per-process virtual address space is enable
9:8	**Reserved** Project: All Format: MBZ

INSTPM—Instruction Parser Mode Register

8	**Memory Sync Enable**	Project:	DevGT +	Format:	U1
	If set, this bit allows the command stream engine to write out the data from the local caches to memory. This bit is valid only with the Sync flush enable				
7	**CONSTANT_BUFFER Surface Address Offset Enable**	Project:	All	Format :	U1
	When this bit is set, the CONSTANT_BUFFER Buffer Starting Address is used as a SurfaceStateOffset. I.e., it serves as an offset from the Surface State Base Address. Accesses will be subject to Surface State bounds checking. When this bit is not set, the CONSTANT_BUFFER Buffer Starting Address is based on bit 6 of the address. No bounds checking will be performed during access. Format = Enable				
6	**CONSTANT_BUFFER Address Offset Disable**	Project:	All	Format :	U1
	When this bit is clear, the CONSTANT_BUFFER Buffer Starting Address is used as a GeneralStateOffset. I.e., it serves as an offset from the General State Base Address. Accesses will be subject to General State bounds checking. When this bit is set, the CONSTANT_BUFFER Buffer Starting Address is used as a true GraphicsAddress (not an offset). No bounds checking will be performed during access. Format = Disable				
5	**Sync Flush Enable**	Project:	All	Format :	U1
	This field is used to request a Sync Flush operation. The device will automatically clear this bit before completing the operation. See Sync Flush (*Programming Environment*). **Programming Note:** • The command parser must be stopped prior to issuing this command by setting the **Stop Rings** bit in register **MI_MODE**. Only after observing **Rings Idle** set in **MI_MODE** can a Sync Flush be issued by setting this bit. Once this bit becomes clear again, indicating flush complete, the command parser is re-enabled by clearing **Stop Rings**. • **Errata:** Sync Flush cannot be used while a media scoreboard kernel is running. Format = Enable (cleared by HW)				
4	**Global Debug Enable**	Project:	All	Format :	U1
	This field is used to enable the debug capability. Setting this bit allows the hardware to start incrementing the registers corresponding to the debug feature. Format = Enable				
3	**Blt Instruction Disable**	Project:	All	Format :	U1
	This bit instructs the Renderer instruction parser to parse and error-check BLT instructions, but not execute them. Format = Disable				

INSTPM—Instruction Parser Mode Register

2	**3D Rendering Instruction Disable** | Project: | All | Format: | U1 This bit instructs the Renderer instruction parser to parse and error-check 3D Rendering instructions, but not execute them. This bit must always be set by software if **3D State Instruction Disable** is set. Setting this bit *without* setting **3D State Instruction Disable** *is* allowed. Format = Disable
1	**3D State Instruction Disable** | Project: | All | Format : | U1 This bit instructs the Renderer instruction parser to parse and error-check 3D State instructions, but not execute them. This bit should *not* be set unless **3D Rendering Instruction Disable** (bit 2) is also set. Format = Disable
0	**Texture Palette Load Instruction Disable** | Project: | All | Format : | U1 This bit instructs the Renderer instruction parser to parse and error-check Texture Palette Load instructions, but not execute them. Format = Disable

8.12.2 Cache_Mode_0— Cache Mode Register 0

Cache_Mode_0— Cache Mode Register 0

Register Type:	MMIO
Address Offset:	2120h
Project:	All
Default Value:	0000 6820h
Access:	R/W
Size (in bits):	32

This register is used to control the operation of the Render and Sampler L2 Caches. All reserved bits are implemented as read/write.

This Register is saved and restored as part of Context.

Bit	Description
31:16	**Masks** Format: | Mask[15:0] A “1” in a bit in this field allows the modification of the corresponding bit in Bits 15:0.

Cache_Mode_0— Cache Mode Register 0

15

Sampler L2 Disable

Project:	All
Default Value:	0h
Format:	Disable

Value	Name	Description	Project
0h		Sampler L2 Cache Enabled.	All
1h		Sampler L2 Cache Disabled all accesses are treated as misses.	All

Errata	Description	Project
BWT012	Setting this bit is UNDEFINED.	DevBW-A,B

14:13

Sampler L2 Page Gathering Fifo Modes

Project:	All
Default Value:	3h
Format:	U3

Value	Name	Description	Project
00		No Page Gathering, No Interleaving.	All
01		On Page Gathering based on Page Size described in Low Priority Grace Period Page Size. No Interleaving.	All
10		Interleaved based on Tile Type and address bits A6, A9 and A10.	All
11		Interleaved on page gathering as combination of modes 1 and 2.	All

Cache_Mode_0— Cache Mode Register 0

12:10

Page Gather Limit	
Project:	All
Default Value:	2h
Format:	U3
Used when bits 14:13 are set to 1 or 3. Determines the maximum number of on page requests gathered.	

Value	Name	Description	Project
000		4 Requests.	All
001		6 Requests.	All
010		8 Requests.	All
011		10 Requests.	All
100		12 Requests.	All
101		14 Requests.	All
110		16 Requests.	All
111		As much as the FIFO allows.	All

9

Sampler L2 TLB Prefetch Enable	
Project:	All
Default Value:	0h
Format:	Enable

Value	Name	Description	Project
0h		TLB Prefetch Disabled	All
1h		TLB Prefetch Enabled	All

8

Reserved	Project:	All	Format:	MBZ

Cache_Mode_0— Cache Mode Register 0

<table>
<tr><td>7:6</td><td>
<table>
<tr><td colspan="2"><b>Sampler L2 Request Arbitration</b></td></tr>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>U2</td></tr>
<tr><td colspan="2"></td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>00</td><td></td><td>Round Robin</td><td>All</td></tr>
<tr><td>01</td><td></td><td>Fetch are Highest Priority</td><td>All</td></tr>
<tr><td>10</td><td></td><td>Constants are Highest Priority</td><td>All</td></tr>
<tr><td>11</td><td></td><td>Reserved</td><td>All</td></tr>
</table>
</td></tr>
<tr><td>5</td><td>
<table>
<tr><td><b>MT Constant Read Bug Fix Disable Chicken Bit</b></td><td>Project:</td><td>DevBW, DevCL</td><td>Format:</td><td>Disable</td></tr>
<tr><td colspan="5">If this bit is set, the MT Constant Read bug fix is disabled. This means constant reads must be routed to the render cache in order to function correctly. Clearing this bit enables the bug fix and allows constant reads to be done via the texture cache. Note that this bit is set by default.</td></tr>
</table>
</td></tr>
<tr><td>4:3</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>2</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>
<table>
<tr><td><b>Disable clock gating in the pixel backend</b></td><td>Project:</td><td>DevCTG</td><td>Format:</td><td>Disable</td></tr>
<tr><td colspan="5">MCL related clock gating is disabled in the pixel backend.</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
</table>

Cache_Mode_0— Cache Mode Register 0

0

Render Cache Operational Flush Enable	
Project:	All
Default Value:	0h
Format:	Enable

Value	**Name**	**Description**	**Project**
0h	Disable	Operational Flush Disabled (recommended for performance when not rendering to the front buffer)	All
1h	Enable	Operational Flush Enabled (required when rendering to the front buffer)	All

Errata	**Description**	**Project**
BWT006	This bit must be clear; Operational Flush cannot be enabled.	DevBW-A,B

8.12.3 Cache_Mode_1— Cache Mode Register 1

Cache_Mode_1— Cache Mode Register 1

Register Type:	MMIO
Address Offset:	2124h
Project:	All
Default Value:	0000 0180h
Access:	Read/32 bit Write
Size (in bits):	32

This Register is saved and restored as part of Context.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:16</td><td>
<table>
<tr><td colspan="2"><b>Mask Bits for 15:0</b></td></tr>
<tr><td>Format:</td><td>Mask[15:0]</td></tr>
<tr><td colspan="2">Must be set to modify corresponding data bit. Reads to this field returns zero.</td></tr>
</table>
</td></tr>
<tr><td>15</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>14</td><td><b>[DevCTG-B3]</b> VFM
1: 3d MarkerFlow fix disabled.
0: 3d MarkerFlow fix enabled</td></tr>
<tr><td>14</td><td>
<table>
<tr><td colspan="3"><b>Enable Pixel Backend Low Precision Float Denorm Handling</b></td></tr>
<tr><td>Project:</td><td colspan="2">HVN/ABD-B0</td></tr>
<tr><td>Default Value:</td><td>0h</td><td></td></tr>
<tr><td>Format:</td><td>U1</td><td></td></tr>
<tr><td colspan="3"></td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td></td><td>Disables preserving denormals through Pixel Backend Blend HW for 16b, 11b and 10b floats.</td><td>HVN/ABD-B0</td></tr>
<tr><td>1</td><td></td><td>Enables preserving denormals through Pixel Backend Blend HW for 16b, 11b and 10b floats.</td><td>HVN/ABD-B0</td></tr>
</table>
</td></tr>
</table>

Cache_Mode_1— Cache Mode Register 1

12

Enable the indirect load of Data through the Vertex Fetch

Project:	All
Default Value:	0h
Format:	U1

Value	Name	Description	Project
0h		Media Object Data transferred through the command streamer	All
1		Media Object Data transferred through the Vertex Fetch	All

11

Instruction and State Cache Invalidate

Project:	All
Default Value:	0h
Format:	U1

When this field is set, all instruction and state caches (level 1 and level 2) are invalidated.

It is intended for debug use. For example, it may be used in conjunction with EU breakpoint control to provide single stepping kernel debugging capability and dynamic breakpoint capability.

Before setting this field, host (debug) software must make sure that the graphics render engine has reached idle state – there is no activity to/from the instruction and state caches. For example, during kernel debug, upon a breakpoint exception, host debug software may delay for a sufficiently long period and then check the EU done signals to make sure that all EUs other than the one(s) causing the breakpoint exception are set. It can then set this field to invalidate the instruction and state caches. This field generates a level control signal. Host software must clear this field, before letting execution to continue (e.g. by clearing the host notification MMIO registers to let the kernel under debug to proceed).

Value	Name	Description	Project
0h		Normal Cache operation.	All
1		Invalidate will be sent to Level 1 and Level 2 caches. (DEBUG ONLY)	All

Cache_Mode_1— Cache Mode Register 1

<table>
<tr><td>10</td><td>

Instruction Level 1 Cache and In-Flight Queue Disable

<table>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>Disable</td></tr>
</table>

<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td></td><td>Cache is enabled.</td><td>All</td></tr>
<tr><td>1h</td><td></td><td>Cache is disabled and all accesses to this cache are treated as misses and sent to L2 cache. Setting this bit overrides the setting of bit 0. (DEBUG ONLY)</td><td>All</td></tr>
</table>

</td></tr>
<tr><td>9</td><td>

Instruction and State Level 2 Cache Fill Buffers Disable

<table>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>Disable</td></tr>
</table>

<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td></td><td>Fill Buffers are enabled.</td><td>All</td></tr>
<tr><td>1h</td><td></td><td>Fill Buffers are disabled. (DEBUG ONLY)</td><td>All</td></tr>
</table>

</td></tr>
</table>

Cache_Mode_1— Cache Mode Register 1

8:7

Sampler Cache Set XOR selection	
Project:	All
Default Value:	3h
Format:	U2

These bits have an impact only when the Sampler cache is configured in 16 way set associative mode. If the cache is being used for immediate data or for blitter data these bits have no effect.

Value	Name	Description	Project
00	Default value	Default behavior to calculate set address, no XOR.	All
01	Scheme 1	New_set_mask[3:0] = Tiled_address[16:13] New_set[3:0] <= New_set_mask[3:0] ^ Old_set[3:0] Rationale: These bits can distinguish among 16 different equivalent classes of virtual pages. These bits also represent the lsb for tile rows ranging from a pitch of 1 tile to 16 tiles.	All

10	Scheme 2	New_set_mask[3] = Tiled_address[17] ^ Tiled_address[16]
		New_set_mask[2] = Tiled_address[16] ^ Tiled_address[15]
		New_set_mask[1] = Tiled_address[15] ^ Tiled_address[14
		New_set_mask[0] = Tiled_address[14] ^ Tiled_address[13]
		New_set[3:0] <= New_set_mask[3:0] ^ Old_set[3:0]
		Rationale: More bits on each XOR can give better statistical uniformity on sets and since two lsbs are taken for each tile row size, it reduces the chance of aliasing on sets.
			All

11	Scheme 3	New_set_mask[3] = Tiled_address[22] ^ Tiled_address[21] ^ Tiled_address[20] ^ Tiled_address[19]
		New_set_mask[2] = Tiled_address[18] ^ Tiled_address[17] ^ Tiled_address[16]
		New_set_mask[1] = Tiled_address[15] ^ Tiled_address[14]
		New_set_mask[0] = Tiled_address[13]
		New_set[3:0] <= New_set_mask[3:0] ^ Old_set[3:0]
		Rationale: More bits on each XOR can give better statistical uniformity on sets and since each XOR has different bits, it reduces the chance of aliasing on sets even more.
			All

<table>
<tr>
<td>6:5</td>
<td>
<table>
<tr><td colspan="3"><b>Data Cache Set XOR Selection</b></td></tr>
<tr><td>Project:</td><td colspan="2">DevCTG+</td></tr>
<tr><td>Default Value:</td><td>0h</td><td></td></tr>
<tr><td>Format:</td><td>U2</td><td></td></tr>
<tr><td colspan="3">These bits have an impact only when the data cache is configured in 16 way set associative mode. If the cache is being used for immediate data or for blitter data these bits have no effect.</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>00</td><td>Default value</td><td>Default behavior to calculate set address, no XOR.</td><td>DevCTG +</td></tr>
<tr><td>01</td><td>Scheme 1</td><td>New_set_mask[1:0] = Tiled_address[14:13]
New_set[1:0] <= New_set_mask[1:0] ^ Old_set[1:0]
Rationale: These bits can distinguish among 16 different equivalent classes of virtual pages. These bits also represent the lsb for tile rows ranging from a pitch of 1 tile to 16 tiles.</td><td>DevCTG +</td></tr>
<tr><td>10</td><td>Scheme 2</td><td>New_set_mask[1] = Tiled_address[15] ^ Tiled_address[14]
New_set_mask[0] = Tiled_address[14] ^ Tiled_address[13]
New_set[1:0] <= New_set_mask[1:0] ^ Old_set[1:0]
Rationale: More bits on each XOR can give better statistical uniformity on sets and since two lsbs are taken for each tile row size, it reduces the chance of aliasing on sets.</td><td>DevCTG +</td></tr>
<tr><td>11</td><td>Scheme 3</td><td>New_set_mask[1] = Tiled_address[15] ^ Tiled_address[14]
New_set_mask[0] = Tiled_address[13]
New_set[1:0] <= New_set_mask[1:0] ^ Old_set[1:0]
Rationale: More bits on each XOR can give better statistical uniformity on sets and since each XOR has different bits, it reduces the chance of aliasing on sets even more.</td><td>DevCTG +</td></tr>
</table>
</td>
</tr>
<tr>
<td>4</td>
<td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr><td>3:2</td><td>
<table>
<tr><td colspan="2"><b>Burst-length and water-mark Control</b>
This bits apply to RCZ and RCC.</td></tr>
<tr><td>Project:</td><td>HVN/ABD-B0</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>U2</td></tr>
<tr><td colspan="2"></td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td></td><td>Water-mark trips at FIFO is half full and burst-length is such that FIFO completely drains OR remains half full after initiating writes. Hence, this setting guarantees a burst equal to half the FIFO size.</td><td>HVN/ABD-B0</td></tr>
<tr><td>1h</td><td></td><td>Water-mark trips at FIFO is full and burst-length is such that FIFO completely drains OR remains full after initiating writes. Hence, this setting guarantees a burst equal to the FIFO size.</td><td>HVN/ABD-B0</td></tr>
<tr><td>2h</td><td></td><td>Not used</td><td>HVN/ABD-B0</td></tr>
<tr><td>3h</td><td></td><td>No water-mark. Hence as long as write-back FIFO is valid, present a write-request.</td><td>HVN/ABD-B0</td></tr>
</table>
</td></tr>
<tr><td>3</td><td>
<table>
<tr><td colspan="2"><b>A-step bug fix bit for rcc allocation</b></td></tr>
<tr><td>Project:</td><td>DevBW-A, DevBW-B</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>U1</td></tr>
<tr><td colspan="2">This bit <u>should always be set</u> for proper operation on BW-A,B</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td></td><td>This bug fix is disabled.</td><td>DevBW-A, DevBW-B</td></tr>
<tr><td>1h</td><td></td><td>Bug fix is active and will solve random pixel corruption issues due to this bug. It slows down allocation to one allocation every 4 clock. In the A and B-steps, 3d performance will not be bottlenecked by this bug fix. Media performance impact will be minor.</td><td>DevBW-A, DevBW-B</td></tr>
</table>
</td></tr>
<tr><td>2</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>1</td><td>
<table>
<tr><td colspan="2"><b>Instruction and State Level 2 Cache Disable</b></td></tr>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>Disable</td></tr>
<tr><td colspan="2"></td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td></td><td>Cache is enabled.</td><td>All</td></tr>
<tr><td>1h</td><td></td><td>Cache is disabled and all accesses to this cache are treated as misses. (DEBUG ONLY)</td><td>All</td></tr>
</table>
</td></tr>
<tr><td>0</td><td>
<table>
<tr><td colspan="2"><b>Instruction Level 1 Cache Disable</b></td></tr>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>Disable</td></tr>
<tr><td colspan="2"></td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td></td><td>Cache is enabled.</td><td>All</td></tr>
<tr><td>1h</td><td></td><td>Cache is disabled and all accesses to this cache are treated as misses, but only requests with unique addresses are sent to the L2. (DEBUG ONLY)</td><td>All</td></tr>
</table>
</td></tr>
</table>

8.12.4 FBC RT BASE ADDRESS REGISTER

FBC_RT_BASE_ADDR_REGISTER

Register Type:	MMIO
Address Offset:	2128h
Project:	HVN/ABD
Default Value:	--
Access:	Read/32 bit Write
Size (in bits):	32

This Register is saved and restored as part of Context.

Bit	Description
31:12	**4KB aligned Base Address as mapped in the PPGTT (in the AS mode) OR in the GGTT (in the single-context scheduling mode) For the render target. This register must be programmed in either multi-context scheduling or single-context scheduling mode. This base address must be the one that is either front buffer or the back-buffer (a flip target). It can be only programmed once per context. It must be programmed before any draw call binding that render target base address.** Format: Base Address[31:12] Must be set to modify corresponding data bit. Reads to this field returns zero.
11:1	**Reserved** Project: All Format: MBZ

8.12.5 BB_ADDR—Batch Buffer Head Pointer Register

<table>
<tr><th colspan="2">BB_ADDR—Batch Buffer Head Pointer Register</th></tr>
<tr><td><b>Register Type:</b></td><td>MMIO</td></tr>
<tr><td><b>Address Offset:</b></td><td>2140h</td></tr>
<tr><td><b>Project:</b></td><td>All</td></tr>
<tr><td><b>Default Value:</b></td><td>0000 0000 0000 0000h</td></tr>
<tr><td><b>Access:</b></td><td>RO</td></tr>
<tr><td><b>Size (in bits):</b></td><td>64</td></tr>
<tr><td colspan="2">This register contains the current DWord Graphics Memory Address of the last-initiated batch buffer.</td></tr>
<tr><td><b>Bit</b></td><td><b>Description</b></td></tr>
<tr><td>63:32</td><td><table><tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>31:2</td><td><table><tr><td><b>Batch Buffer Head Pointer</b></td><td>Project:</td><td>All</td><td>Format:</td><td>GraphicsAddress[31:2]</td></tr><tr><td colspan="5">This field specifies the DWord-aligned Graphics Memory Address where the last initiated Batch Buffer is currently fetching commands. If no batch buffer is currently active, the Valid bit will be 0 and this field will be meaningless.</td></tr></table></td></tr>
<tr><td>1</td><td><table><tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>0</td><td><table><tr><td colspan="2"><b>Valid</b></td></tr><tr><td>Project:</td><td>All</td></tr><tr><td>Default Value:</td><td>0h</td></tr><tr><td>Format:</td><td>U1</td></tr><tr><td colspan="2"></td></tr></table><table><tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr><tr><td>0h</td><td>Invalid</td><td>Batch buffer Invalid</td><td>All</td></tr><tr><td>1h</td><td>Valid</td><td>Batch buffer Valid</td><td>All</td></tr></table></td></tr>
</table>

8.12.6 BB_STATE – Batch Buffer State Register

BB_STATE – Batch Buffer State Register

Register Type:	MMIO
Address Offset:	2110h
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

This register contains the attributes of the last batch buffer initiated from the Ring Buffer. These include the memory space select and security indicator.

This register should *not* be written by software. These fields should only get written by a context restore. Software should always set these fields via the MI_BATCH_BUFFER_START command when initiating a batch buffer.

This register is saved and restored with context.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:6</td><td><table><tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>5</td><td><table>
<tr><td colspan="2"><b>Buffer Security Indicator</b></td></tr>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>MI_BufferSecurityType</td></tr>
<tr><td colspan="2">If set, this batch buffer is non-secure and cannot execute privileged commands nor access privileged (GGTT) memory. It will be accessed via the PPGTT. If clear, this batch buffer is secure and will be accessed via the GGTT.

Note: This field reflects the effective security level and may not be the same as the Buffer Security Indicator written using MI_BATCH_BUFFER_START.</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>MIBUFFER_SECURE</td><td>Located in GGTT memory</td><td>All</td></tr>
<tr><td>1h</td><td>MIBUFFER_NONSECURE</td><td>Located in PPGTT memory</td><td>All</td></tr>
</table></td></tr>
<tr><td>4:0</td><td><table><tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

8.12.7 CTXT_SR_CTL – Context Save / Restore Control Register

CTXT_SR_CTL – Context Save/Restore Control Register

Register Type:	MMIO
Address Offset:	2714h
Project:	All
Default Value:	0000 0000h
Access:	R/W (Debug Only)
Size (in bits):	32

This register is saved and restored with context.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:2</td><td><table><tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>1</td><td><table><tr><td colspan="2"><b>Extended Context Enable</b></td></tr><tr><td>Project:</td><td>All</td></tr><tr><td>Default Value:</td><td>0h</td></tr><tr><td>Format:</td><td>Enable</td></tr></table>
If this bit is set, the extended portion of the render context will be saved and restored with the current context. If clear, extended context will not be a part of this context. Note that since this register is part of ring context, each context can have its own setting for this bit. Extended context can thus be selected on a per-context basis. Note that extended context is part of render context, so that if Render Context Restore Inhibit is set in the context image, extended context will not be restored (the first time) even if this bit is set.
<table><tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr><tr><td>0h</td><td>Disable</td><td>The current context does not include extended context</td><td>All</td></tr><tr><td>1h</td><td>Enable</td><td>The current context does include extended context.</td><td>All</td></tr></table></td></tr>
<tr><td>0</td><td><table><tr><td><b>Render Context Restore Inhibit</b></td><td>Project:</td><td>All</td><td>Format:</td><td>U1</td></tr><tr><td colspan="5">This is not a true register bit. This bit should be set in the context image of a ring context that is being submitted for the first time. Setting this bit will inhibit the restoring of render context (including extended context if applicable) so that restoring of an uninitialized render context can be prevented. This bit will always be set on a context save (since the render context cannot be uninitialized on context save – it will always contain at least default values.)</td></tr></table></td></tr>
</table>

8.13 Logical Context Support

8.13.1 CCID—Current Context ID Register

CCID—Current Context ID Register

Register Type:	MMIO
Address Offset:	2180h
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

This register contains the current "logical rendering context address" associated with the ring buffer.

Programming Note: The CCID register must not be written directly (via MMIO) unless the Command Streamer is completely idle (i.e., the Ring Buffer is empty and the pipeline is idle). Note that, under normal conditions, the CCID register should only be updated from the command stream using the MI_SET_CONTEXT command.

Bit	Description
31:11	**Logical Render Context Address (LRCA)** Project: All Default Value: 0h Address: GraphicsAddress[31:11] This field contains the 4 KB-aligned Graphics Memory Address of the current Logical Rendering Context. Bit 11 MBZ. It will point to a Logical Pipeline Context (a subset of a Logical Rendering Context) if loaded using MI_SET_CONTEXT. If this register was set using MI_SET_CONTEXT with the **Memory Space Select** set to Physical Main Memory, this field contains the 2 KB-aligned "Effective Local Memory" *physical* Main Memory address of the current Logical Pipeline Context.
10:8	**Reserved** Project: All Format: MBZ
7:4	**Physical Start Address Extension** Project: All Default Value: 0h Address: GraphicsAddress[35:32] This field specifies Bits 35:32 of the starting *physical* address *if* **Memory Space Select** of the last MI_SET_CONTEXT command was set to Physical Main Memory.
3	**Extended State Save** Project: All Format: Enable

CCID—Current Context ID Register

<table>
<tr><td></td><td>
<table>
<tr><td><b>Enable</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="5">If set, the extended state identified in the Logical Context Data section of the Memory Data Formats chapter, is saved as part of switching away from this logical context.</td></tr>
</table>
</td></tr>
<tr><td>2</td><td>
<table>
<tr><td><b>Extended State Restore Enable</b></td><td>Project:</td><td>All</td><td>Format:</td><td>Enable</td></tr>
<tr><td colspan="5">If set, the extended state identified in the Logical Context Data section of the Memory Data Formats chapter, was loaded (or restored) as part of switching to this logical context.</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>0</td><td>
<table>
<tr><td colspan="2"><b>Valid</b></td></tr>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>U1</td></tr>
<tr><td colspan="2"></td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>Invalid</td><td>The other fields of this register are invalid. A switch away from the context will not invoke a context save operation.</td><td>All</td></tr>
<tr><td>1h</td><td>Valid</td><td>The other fields of this register are valid, and a switch from the context will invoke the normal context save/restore operations.</td><td>All</td></tr>
</table>
</td></tr>
</table>

8.13.2 CXT_SIZE—Context Size with Extended State

CXT_SIZE—Context Size with Extended State

Register Type:	MMIO
Address Offset:	21A0h
Project:	All
Default Value:	0000 0013h
Access:	Read/32 bit Write
Size (in bits):	32

Bit	Description
31:5	**Reserved** Project: All Format: MBZ
4:0	**Size** Project: All Default Value: 13h Format: U5-1 Size of pipeline logical rendering context including the extended state in 64B quantities minus one.

8.13.3 CXT_SIZE_NOEXT—Context Size without the Extended State

CXT_SIZE_NOEXT—Context Size without the Extended State

Register Type:	MMIO
Address Offset:	21A4h
Project:	All
Default Value:	0000 000Fh
Access:	Read/32 bit Write
Size (in bits):	32

Bit	Description
31:5	**Reserved** Project: All Format: MBZ

CXT_SIZE_NOEXT—Context Size without the Extended State		
4:0	**Size**	
	Project:	All
	Default Value:	Fh
	Format:	U5-1
	Size of pipeline logical rendering context excluding the extended state in 64B quantities minus one.	

8.14 Arbitration Control, and Scratch Bits

8.14.1 MI_DISPLAY_POWER_DOWN—Display Power Down ([DevCL+])

MI_DISPLAY_POWER_DOWN—Display Power Down	
Register Type:	MMIO
Address Offset:	20E0h
Project:	DevCL+
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

The MI_DISPLAY_POWER_DOWN register contains the Display Power Down Enable bit which is used to enable display power down prior to entering C3SR state.

This Register is NOT saved and restored as part of Context.

Bit	Description							
31:16	**Reserved**	Project:	DevCL	Format:	MBZ			
15	**Display Power Down Enable**	Project:	DevCL	Format:	Enable			
	The bit enables the chipset to put the DIMMs in self refresh when the display conditions are right (No VGA or Overlay, only 1 display pipe enabled) and the CPU is in the C3+ state. Note that setting this bit is not required for DIMMs to enter self-refresh for any device state higher than D0.							
14:0	**Reserved**	Project:	DevCL	Format:	MBZ			

8.14.2 MI_ARB_STATE—Memory Interface Arbitration State Register

<table>
<tr><th colspan="2">MI_ARB_STATE—Memory Interface Arbitration State Register</th></tr>
<tr><td><b>Register Type:</b></td><td>MMIO</td></tr>
<tr><td><b>Address Offset:</b></td><td>20E4h</td></tr>
<tr><td><b>Project:</b></td><td>All</td></tr>
<tr><td><b>Default Value:</b></td><td>0000 0040h</td></tr>
<tr><td><b>Access:</b></td><td>R/W</td></tr>
<tr><td><b>Size (in bits):</b></td><td>32</td></tr>
<tr><td colspan="2">The MI_ARB_STATE register contains state information that controls arbitration aspects of the Memory Interface function.</td></tr>
</table>

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:16</td><td>
<table>
<tr><td colspan="2"><b>Mask Bits</b></td></tr>
<tr><td>Format:</td><td>Mask[15:0]</td></tr>
<tr><td colspan="2">Must be set to modify corresponding bit in Bits 15:0. (All bits implemented)</td></tr>
</table>
</td></tr>
<tr><td>15</td><td>
<table>
<tr><td colspan="2"><b>Render/Sampler TLB Request Priority</b></td></tr>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>U1</td></tr>
<tr><td colspan="2"></td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td></td><td>TLBs are above the corresponding data requests in priority. That is Render TLB fetch is above Render reads and writes, Sampler TLB fetches are above Sampler Fetches. This is the default setting and used for normal operation.</td><td>All</td></tr>
<tr><td>1h</td><td></td><td>TLBs are at the lowest priority (above FBC) with Sampler TLB fetches higher than render.</td><td>All</td></tr>
</table>
</td></tr>
<tr><td>14:9</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
<tr><td colspan="5">Read/Write (SW must maintain setting)</td></tr>
</table>
</td></tr>
</table>

MI_ARB_STATE—Memory Interface Arbitration State Register

8

Suppress Cacheable indicator from Render Command Stream write requests		
Project:	All	
Default Value:	0h	
Format:	U1	

Value	Name	Description	Project
0h	Snooped	Cacheable write cycles from Render Command Stream are snooped on the FSB.	All
1h	Non-Snooped	Cacheable write cycles from Render Command Stream are not snooped on the FSB. These writes are processed as non-snoop.	All

Errata	Description	Project
BWT010	Setting this bit may cause UNDEFINED behavior (extra cycles issued to different addresses in addition to the specified address.)	DevBW-A

MI_ARB_STATE—Memory Interface Arbitration State Register

7:5

Time Slice	
Project:	All
Default Value:	2h
Format:	U3

Applicable to Render Cache, Sampler Cache, Pixel Shader, Frame Buffer, Command stream and Host Requests. Time Slice is fixed at 1 for TLB and snoop requests, and not applicable to Isochronous Streams.

The (value programmed –1) determines the number of Page Hits before arbitration switch for a low priority stream interrupted by a higher priority stream as long as the lower priority stream is active.

If set to '000' the arbiter does apply a page hit grace period.

In 64B Requests

Value	**Name**	**Description**	**Project**
000	1 Request	1 Requests (This setting implies that the grace period is disabled)	All
001	2 Requests	2 Requests	All
010	4 Requests	4 Requests	All
011	6 Requests	6 Requests	All
100	8 Requests	8 Requests	All
101	10 Requests	10 Requests	All
110	14 Requests	14 Requests	All
111	16 Requests	16 Requests	All

4

Low Priority Grace Period Page Size	
Project:	All
Default Value:	0h
Format:	U1

Value	**Name**	**Description**	**Project**
0h		Grace period on-page indicator uses 4KB pages in the command streams and caches. (Default)	All
1h		Grace period on-page indicator uses 8KB pages in the command streams and caches.	All

3

Reserved	Project:	All	Format:	MBZ

Read/Write (SW must maintain setting)

MI_ARB_STATE—Memory Interface Arbitration State Register

2

Display A/B Trickle Feed Disable	
Project:	All
Default Value:	0h
Format:	Disable

Value	Name	Description	Project
0h		Enable	All
1h		Disable (Turn off trickle feed Display request)	All

Programming Notes	Project
For mobile devices ([DevCL]), this bit should always be *set* to disable trickle feed.	DevCL
[DevBW] must always set to disable trickle feed	DevBW

1

Reserved	Project:	All	Format:	MBZ
Read/Write (SW must maintain setting)				

0

Display A/B Priority Select	
Project:	All
Default Value:	0h
Format:	U1

This bit determines the arbitration priority of accesses among the high priority streams.

Value	Name	Description	Project
0h	DA/DB/Others	Set this when Display Plane A is the Primary	All
1h	DB/DA/Others	Set this when Display Plane B is the Primary	All

8.14.3 MI_RDRET_STATE—Memory Interface Read Return State Register

<table>
<tr><th colspan="2">MI_RDRET_STATE—Memory Interface Read Return State Register</th></tr>
<tr><td>Register Type:</td><td>MMIO</td></tr>
<tr><td>Address Offset:</td><td>20FCh [DevCL] 20E0h [DevBW]</td></tr>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Default Value:</td><td>0000 0000h</td></tr>
<tr><td>Access:</td><td>R/W</td></tr>
<tr><td>Size (in bits):</td><td>32</td></tr>
<tr><td colspan="2">The MI_RDRET_STATE register contains state information that controls data return aspects of the Memory Interface function. This register is used strictly for HVM testing. Any functional usage of this register is undefined. None of the TLB read returns from memory are impacted by this register.
This Register is NOT saved and restored as part of Context.</td></tr>
</table>

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:16</td><td><table>
<tr><td colspan="2">Mask Bits</td></tr>
<tr><td>Format:</td><td>Mask[15:0]</td></tr>
<tr><td colspan="2">Must be set to a 1 to allow modification of corresponding bit in Bits 15:0. (All implemented bits)</td></tr>
</table></td></tr>
<tr><td>15</td><td><table>
<tr><td>HVM Enable Bit</td><td>Project:</td><td>All</td><td>Format:</td><td>Enable</td></tr>
<tr><td colspan="5">This bit must be set to ‘1’ to enable HVM loopback mode and enable random internal data returns from CI. This bit must be programmed after the other client specific bits are programmed to desired values.</td></tr>
</table></td></tr>
<tr><td>14:9</td><td><table>
<tr><td>Reserved</td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td>8</td><td><table>
<tr><td colspan="2">Vertex Fetch Cache select</td></tr>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>U1</td></tr>
<tr><td colspan="2">This bit determines the read return for Vertex Fetch Reads</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td></td><td>Return Data from memory</td><td>All</td></tr>
<tr><td>1h</td><td></td><td>Return data from a random data generator on-chip</td><td>All</td></tr>
</table></td></tr>
<tr><td>7</td><td><table>
<tr><td>Reserved</td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
<tr><td colspan="5">Was Read Only Cache select</td></tr>
</table></td></tr>
</table>

MI_RDRET_STATE—Memory Interface Read Return State Register

6

Display Sprite B Read Return Select	
Project:	DevCTG
Default Value:	0h
Format:	U1
This bit determines the read return for Display Sprite B Reads	

Value	Name	Description	Project
0h		Return Data from memory	DevCTG
1h		Return data from a random data generator on-chip	DevCTG

6

Overlay return Select	
Project:	DevCL
Default Value:	0h
Format:	U1
This bit determines the read return for Overlay streamer Reads	

Value	Name	Description	Project
0h		Return Data from memory	DevCL
1h		Return data from a random data generator on-chip	DevCL

6

Reserved	Project:	DevBW	Format:	MBZ

5

Color / Z return Select	
Project:	All
Default Value:	0h
Format:	U1
This bit determines the read return for Low Priority Reads	

Value	Name	Description	Project
0h		Return Data from memory	All
1h		Return data from a random data generator on-chip	All

MI_RDRET_STATE—Memory Interface Read Return State Register

4

Sampler Read return Select	
Project:	All
Default Value:	0h
Format:	U1
This bit determines the read return for Low Priority Reads	

Value	**Name**	**Description**	**Project**
0h		Return Data from memory	All
1h		Return data from a random data generator on-chip	All

3

Cursor (A and B) Read return Select	
Project:	All
Default Value:	0h
Format:	U1
This bit determines the read return for Display Reads	

Value	**Name**	**Description**	**Project**
0h		Return Data from memory	All
1h		Return data from a random data generator on-chip	All

2

Display Sprite A Read Return Select	
Project:	DevCTG
Default Value:	0h
Format:	U1
This bit determines the read return for Display Sprite A Reads	

Value	**Name**	**Description**	**Project**
0h		Return Data from memory	DevCTG
1h		Return data from a random data generator on-chip	DevCTG

MI_RDRET_STATE—Memory Interface Read Return State Register

2

Display C Read return Select	
Project:	All
Default Value:	0h
Format:	U1
This bit determines the read return for Display C Reads	

Value	Name	Description	Project
0h		Return Data from memory	All
1h		Return data from a random data generator on-chip	All

1

Display B Read return Select	
Project:	All
Default Value:	0h
Format:	U1
This bit determines the read return for Display Reads	

Value	Name	Description	Project
0h		Return Data from memory	All
1h		Return data from a random data generator on-chip	All

0

Display A Read return Select	
Project:	All
Default Value:	0h
Format:	U1
This bit determines the read return for Display Reads	

Value	Name	Description	Project
0h		Return Data from memory	All
1h		Return data from a random data generator on-chip	All

8.14.4 MI_MODE — Mode Register for Software Interface

MI_MODE — Mode Register for Software Interface

Register Type:	MMIO
Address Offset:	209Ch
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32

The MI_MODE register contains information that controls software interface aspects of the Memory Interface function.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:16</td><td>
<table>
<tr><td colspan="2"><b>Masks</b></td></tr>
<tr><td>Format:</td><td>Mask[15:0]</td></tr>
<tr><td colspan="2">A “1” in a bit in this field allows the modification of the corresponding bit in Bits 15:0</td></tr>
</table>
</td></tr>
<tr><td>15</td><td>
<table>
<tr><td colspan="2"><b>Suspend Flush</b></td></tr>
<tr><td>Project:</td><td>DevCTG+</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>U1</td></tr>
<tr><td colspan="2"></td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>Delay Flush</td><td>HW will delay the flush because of sync flush or VTD regimes until reset</td><td>DevCTG</td></tr>
<tr><td>1h</td><td>No Delay</td><td>HW will not delay flush</td><td>DevCTG</td></tr>
<tr><td>0h</td><td>Delay Flush</td><td>HW will delay the flush because of sync flush or VTD regimes until reset, this bit will get set by MI_SUSPEND_FLUSH as well</td><td>HVN/ABD</td></tr>
<tr><td>1h</td><td>No Delay</td><td>HW will not delay flush, this bit will get set by MI_SUSPEND_FLUSH as well</td><td>HVN/ABD</td></tr>
</table>
</td></tr>
</table>

MI_MODE — Mode Register for Software Interface

<table>
<tr><td>14</td><td>
<table>
<tr><td colspan="2"><b>Async Flip Performance mode</b></td></tr>
<tr><td>Project:</td><td>DevCTG+</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>U1</td></tr>
<tr><td colspan="2"></td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>Performance mode enabled</td><td>The stall of the flip event is in the windower</td><td>DevCTG</td></tr>
<tr><td>1h</td><td>Performance mode disabled</td><td>The stall of the flip event is in the command stream</td><td>DevCTG</td></tr>
</table>
</td></tr>
<tr><td>13</td><td>
<table>
<tr><td colspan="2"><b>Flush Performance mode</b></td></tr>
<tr><td>Project:</td><td>DevCL, DevCTG+</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>U1</td></tr>
<tr><td colspan="2"></td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>run fast restore</td><td>No NonPipelined SV flush.</td><td>DevCL, DevCTG</td></tr>
<tr><td>1h</td><td>run slow legacy restore</td><td>With NonPipelined SV flush.</td><td>DevCL, DevCTG</td></tr>
</table>
</td></tr>
<tr><td>13</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>DevBW</td><td>Format:</td><td>MBZ</td></tr>
<tr><td colspan="5">Read/Write</td></tr>
</table>
</td></tr>
<tr><td>12</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>11</td><td>
<table>
<tr><td><b>Invalidate UHPTR enable</b></td><td>Project:</td><td>All</td><td>Format:</td><td>Enable</td></tr>
<tr><td colspan="5">If bit set H/W clears the valid bit of UHPTR (2134h, bit 0) when current active head pointer is equal to UHPTR.</td></tr>
</table>
</td></tr>
<tr><td>10</td><td>
<table>
<tr><td><b>Power of 2 Fences Enable</b></td><td>Project:</td><td>All</td><td>Format:</td><td>Enable</td></tr>
<tr><td colspan="5">This field is used to indicate to the hardware that the fences in use currently are for Power of 2 tile pitch. This bit is used by the chipset for performance enhancement.</td></tr>
</table>
</td></tr>
</table>

MI_MODE — Mode Register for Software Interface

9

Rings Idle		
Project:	All	
Default Value:	0h	
Format:	U1	
Read Only Status bit		

Value	Name	Description	Project
0h	Not Idle	Parser not Idle or Ring Arbiter not Idle.	All
1h	Idle	Parser Idle and Ring Arbiter Idle.	All

Programming Notes	Project
Writes to this bit are not allowed.	All

8

Stop Rings		
Project:	All	
Default Value:	0h	
Format:	U1	

Value	Name	Description	Project
0h		Normal Operation.	All
1h		Parser is turned off and Ring arbitration is turned off.	All

Programming Notes	Project
Software must set this bit to force the Rings and Command Parser to Idle. Software must read a “1” in Ring Idle bit after setting this bit to ensure that the hardware is idle.	All
Software must clear this bit for Rings to resume normal operation.	All

MI_MODE — Mode Register for Software Interface

7

Vertex Shader Cache Mode

Project:	All
Default Value:	0h
Format:	U1

Value	Name	Description	Project
0h	Non-LRA	Non-LRA mode of allocation. Vertex shader cache is allocated on the basis of the reference count of individual vertices	All
1h	LRA	LRA mode of allocation. Used for validation purposes.	All

6

Vertex Shader Timer Dispatch Enable

Project:	All
Default Value:	0h
Format:	Enable

Value	Name	Description	Project
0h	Disable	Disable the timer for dispatch of single vertices from the vertex shader. Vertex shader will try to collect 2 vertices before a dispatch	All
1h	Enable	Enable the timer for dispatch of single vertices. Dispatch a single vertex shader thread after the timer expires.	All

Programming Notes	Project
To avoid deadlock conditions in hardware this bit needs to be set for normal operation.	All

5

FBC2 Modification Enable

Project:	All
Default Value:	0h
Format:	Enable

Value	Name	Description	Project
0h	Disable	FBC logic does not look at the modifications to the frame buffer.	All
1h	Enable	FBC logic looks at the modifications into the frame buffer.	All

MI_MODE — Mode Register for Software Interface

4

Reserved.

3

Physical Batch Buffer 4K size limit disable (test mode)		
Project:	All	
Default Value:	0h	
Format:	Disable	

Value	Name	Description	Project
0h	Disable	Physical batch buffers more than 4K in size are not permitted.	All
1h	Enable	Physical batch buffers more than 4K in size are permitted.	All

2

Pipe control depth in the command streamer		
Project:	DevCTG-B	
Default Value:	0h	
Format:	U1	

Value	Name	Description	Project
0h	8 Pipe Controls	8 pipe controls are supported in the command streamer	DevCTG-B
1h	4 Pipe Controls	4 pipe controls are supported in the command streamer	DevCTG-B

1

Dummy Read Disable	Project:	All	Format:	Disable
Nominally a command stream flush is completed with a dummy read to memory to push all pending writes. Setting this bit to a “1” disables the dummy read.				

0

Mask IIR disable	Project:	All	Format:	Disable
Mask IIR disable. Nominally the Interrupt controller masks interrupts in the IIR register if an interrupt acknowledge from the 3gio interface is pending. Setting this bit to a “1” allows interrupts to be visible to the interrupt controller while an interrupt acknowledge is pending.				

8.14.5 ECOSKPD—ECO Scratch Pad (DEBUG)

<table>
<tr><th colspan="2">ECOSKPD—ECO Scratch Pad (DEBUG)</th></tr>
<tr><td><b>Register Type:</b></td><td>MMIO</td></tr>
<tr><td><b>Address Offset:</b></td><td>21D0h</td></tr>
<tr><td><b>Project:</b></td><td>All</td></tr>
<tr><td><b>Default Value:</b></td><td>00000307h</td></tr>
<tr><td><b>Access:</b></td><td>R/W</td></tr>
<tr><td><b>Size (in bits):</b></td><td>32</td></tr>
<tr><td><b>Trusted Type:</b></td><td>1</td></tr>
<tr><td colspan="2"></td></tr>
</table>

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:16</td><td>
<table>
<tr><th colspan="3">Mask Bits</th></tr>
<tr><td>Format:</td><td>Mask[15:0]</td><td></td></tr>
<tr><td colspan="3">Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)</td></tr>
</table>
</td></tr>
<tr><td>15</td><td><b>Reserved</b> Project: All Format: MBZ</td></tr>
<tr><td>14</td><td>
<table>
<tr><th colspan="3">Vertex Shader Dual dispatch disable</th></tr>
<tr><td>Project:</td><td colspan="2">DevBW-E</td></tr>
<tr><td>Security:</td><td colspan="2">None</td></tr>
<tr><td>Default Value:</td><td>0h</td><td>Enable the dual dispatch</td></tr>
<tr><td>Mask:</td><td colspan="2">MMIO(0x21D0)#30</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>Enable</td><td>HW implements the fix for the enhanced dual dispatch. Dual dispatch is triggered only when the top entry in the tracking FIFO is a</td><td>DevBW-E</td></tr>
<tr><td>1h</td><td>Disable</td><td>Disable the HW fix for the enhanced dual dispatch. Vertex shader will be dispatched as a single vertex everytime the tracking FIF becomes full.</td><td>DevBW-E</td></tr>
</table>
</td></tr>
</table>

ECOSKPD—ECO Scratch Pad (DEBUG)

13

Clipper Performance Fix Disable		
Project:	DevBW-E	
Security:	None	
Default Value:	0h	DefaultVaueDesc
Mask:	MMIO(0x21D0)#29	

Value	Name	Description	Project
0h	Disable	Desc	All
1h	Enable	Desc	All

12

Clipper software workaround for DX10 Enable		
Project:	DevBW-E	
Security:	None	
Default Value:	0h	DefaultVaueDesc
Mask:	MMIO(0x21D0)#28	

Value	Name	Description	Project
0h	Disable	Desc	DevBW-E
1h	Enable	Desc	DevBW-E

Programming Notes	Project
Notes This bit is expected to be used with bit 9 in this register: bit9 bit12 0 0 PerformanceEC0. No software workaround in vs0. 1 0 Not valid. 0 1 Software workaround for Dx10 1 1 No ECO. Will need Software workaround for Dx9.	DevBW-E

11

PL Unit bug fix	Project:	All	Format:	U1
Unspecified ECO disable in the PL unit				

10

RCC Unit Bug fix	Project:	All	Format:	U1
Unspecified ECO disable in the RCC unit				

ECOSKPD—ECO Scratch Pad (DEBUG)

9

Clipper fix for definition of Bad vertex		
Project:	DevCL, DevBW-E	
Security:	None	
Default Value:	0h	DefaultVaueDesc
Mask:	MMIO(0x21D0)#25	

Value	**Name**	**Description**	**Project**
0h	Disable	BAD vertex is dealt as a trivial reject	DevCL, DevBW-E
1h	Enable	BAD vertex is dealt as a must clip instead of trivial reject	DevCL, DevBW-E

8

Clock gating for the RCC (Disable one clock gate cell)		
Project:	DevCL	
Default Value:	0h	DefaultVaueDesc
Format:	U1	FormatDesc

0 = Disable Clock gating

1 = Enable clock gating

7

Clock gating for the MAWB		
Project:	DevCL	
Default Value:	0h	DefaultVaueDesc
Format:	U1	FormatDesc

0 = Disable Clock gating

1 = Enable clock gating

6

Reserved	Project:	All	Format:	MBZ

5

ECO for the VFEunit. Fixes flush between commands		
Project:	DevCTG	
Default Value:	0h	DefaultVaueDesc
Format:	U1	FormatDesc

4

Constant Buffer Save/Restore Disable		
Project:	DevBW-C1+	
Default Value:	0h	DefaultVaueDesc
Format:	U1	FormatDesc

“0” : constant buffer should part of context save/restore

“1”: constant buffer should not be part of context save/restore

ECOSKPD—ECO Scratch Pad (DEBUG)

<table>
<tr><td>3</td><td>
<table>
<tr><td colspan="4"><b>WIZunit Scratch Space ECO</b></td></tr>
<tr><td>Project:</td><td colspan="3">DevBW-C+</td></tr>
<tr><td>Default Value:</td><td>0h</td><td colspan="2">DefaultVaueDesc</td></tr>
<tr><td>Format:</td><td colspan="2">U1</td><td>FormatDesc</td></tr>
<tr><td colspan="4">Enable ECO: Max scratch space (indicated by <b>Per Thread Scratch Space</b> set to 11) is 256KB. 256KB scratch base must be 8M aligned.
Disable ECO: Max scratch space is 12KB.</td></tr>
</table>
</td></tr>
<tr><td>2:0</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
</table>

8.15 Debug Registers

These registers are used to reflect internal hardware state. The intention is to be used for silicon debug

8.15.1 CSFLFSM — Flush FSM (Debug)

CSFLFSM — Flush FSM (Debug)

Register Type:	MMIO
Address Offset:	2200h
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

Bit	Description
31:16	**Reserved: 0x0** Project: All Format: MBZ
15:13	Project: All Format: U3 “000” * (CSFLSHFIFOIDLE_s == ‘1’) + “001” * (CSFLSHFIFOVIRXPHY_s == ‘1’) + “010” * (CSFLSHFIFOWT4ACK_s == ‘1’) + “011” * (CSFLSHFIFOLDSTDW_s == ‘1’) + “100” * (CSFLSHFIFOISCFLUSH_s == ‘1’) + “101” * (CSFLSHFIFOMSI_s == ‘1’) + “110” * (CSFLSHFIFODMYRD_s == ‘1’) + “111”
12:10	Project: All Format: U3 “000” * (CS3DCNTRLIDLE_s == ‘1’) + “001” * (CS3DCNTRLDW1_s == ‘1’) + “010” * (CS3DCNTRLDW2_s == ‘1’) + “011” * (CS3DCNTRLDFIFO_s == ‘1’) + “100” * (CS3DCNTRLWT4DONE_s == ‘1’) + “101” * (CS3DCNTRLNULL_s == ‘1’) + “111”
9:8	Project: All Format: U2 “00” * (URBIDLE_s == ‘1’) + “01” * (URBPIPESEL_s == ‘1’) + “10” * (URBCURBECLEAR_s == ‘1’) + “11” * (URBDEALLOC_s == ‘1’)

CSFLFSM — Flush FSM (Debug)

7:4		Project:	All	Format:	U4
	“0000” * (URBNIDLE_s == ‘1’) + “0001” * (URBNCLR_s == ‘1’) + “0010” * (URBNCLRS_s == ‘1’) + “0011” * (URBNSET_s == ‘1’) + “0100” * (URBNRPLC_s == ‘1’) + “0101” * (URBNRPLC_W_s == ‘1’) + “0110” * (URBCLRWT_s == ‘1’) + “0111” * (URBNPRIM_s == ‘1’) + “1000” * (URBNRPLC_WVS0_s == ‘1’) + “1111”				
3:0		Project:	All	Format:	U4
	“0000” * (IDLE_S == ‘1’)+ “0001” * (NF3DADDR_S == ‘1’)+ “0010” * (NF3DADDR_URB_S == ‘1’)+ “0011” * (NFNPRIM_URBCLR_S == ‘1’)+ “0100” * (NFMDADDR_S == ‘1’)+ “0101” * (NF3DNPRIM_S == ‘1’)+ “0110” * (NFMDNPRIM_S == ‘1’)+ “0111” * (NFURBNPRIM_S == ‘1’)+ “1000” * (NFURBWALLOC_S == ‘1’)+ “1111”				

8.15.2 CSFLFLAG — Flush FLAG (Debug)

<table>
<tr><th colspan="7">CSFLFLAG — Flush FLAG (Debug)</th></tr>
<tr><td><b>Register Type:</b></td><td colspan="6">MMIO</td></tr>
<tr><td><b>Address Offset:</b></td><td colspan="6">2204h</td></tr>
<tr><td><b>Project:</b></td><td colspan="6">All</td></tr>
<tr><td><b>Default Value:</b></td><td colspan="6">0000 0000h</td></tr>
<tr><td><b>Access:</b></td><td colspan="6">R/W</td></tr>
<tr><td><b>Size (in bits):</b></td><td colspan="6">32</td></tr>
<tr><td colspan="7"></td></tr>
<tr><td><b>Bit</b></td><td colspan="6"><b>Description</b></td></tr>
<tr><td rowspan="1">31:17</td><td><b>Reserved: 0x0</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td><td></td></tr>
<tr><td rowspan="2">16:9</td><td></td><td>Project:</td><td>All</td><td>Format:</td><td>U8</td><td></td></tr>
<tr><td colspan="6">csprsrallflsh& csctxlcflsh& csynclcflush &fi_write & fi_depth & fi_timestamp &
fi_iscflush & fi_globalcnt_rst</td></tr>
<tr><td rowspan="2">8</td><td><b>cs_media_select</b></td><td>Project:</td><td>All</td><td>Format:</td><td>U1</td><td></td></tr>
<tr><td colspan="6"></td></tr>
<tr><td rowspan="2">7:0</td><td></td><td>Project:</td><td>All</td><td>Format:</td><td>U8</td><td></td></tr>
<tr><td colspan="6">fi_MURB_chng & fi_MSP_flag &
fi_URB_chng & fi_PSP_flag & fi_BTP_flag &
fi_curbe_opcodes & fi_only_one_curbe_avail &
cs_curbe_set</td></tr>
</table>

8.15.3 CSFLTRK — Flush Track (Debug)

CSFLTRK — Flush Track (Debug)	
Register Type:	MMIO
Address Offset:	2208h
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

Bit	Description
31:13	**Reserved: 0x0** Project: All Format: MBZ
12:8	Project: All Format: U5 fi_3dcntrl_ldfifo & fi_3dcntrlfifo_full & fi_3dcntrl_ram_wren & fi_3dcntrl_ram_wraddr[1:0]
7:0	Project: All Format: U8 fi_3dcntrl_rdptr[1:0] & fi_fiford & fi_3dcntrl_ramwrptr[1:0] & fi_3dcntrl_completeptr_crb2clk[2:0]

8.15.4 CSCMDOP — Instruction DWORD (Debug)

CSCMDOP — Instruction DWORD (Debug)	
Register Type:	MMIO
Address Offset:	220Ch
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

Bit	Description
31:0	**Command Buffer Data** Project: All Format: U32 This field represents the data being parsed by the command streamer currently

8.15.5 CSCMDVLD — Instruction DWORD Valid (Debug)

CSCMDVLD — Instruction DWORD Valid (Debug)

Register Type:	MMIO
Address Offset:	2210h
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

Bit	Description				
31:1	**Reserved**	Project:	All	Format:	MBZ
0	**Command Buffer Valid**	Project:	All	Format:	U1
	Command buffer currently has valid data				

8.15.6 PREEMPTDLY — Power Context Register Address ([DevCTG] Only) (Debug)

PREEMPTDLY — Power Context Register Address (Debug)

Register Type:	MMIO
Address Offset:	2214h
Project:	DevCTG
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

The PREEMPTDLY register contains a delay field which specifies the minimum number of microseconds allowed between honoring preemptions.

A new Run List submission will not be honored (will be internally delayed) until the time from the last one is greater than the delay specified at bits [9:0].

A default value of 0, means that by default, there is no restriction to the time between preemptions.

Bit	Description				
31:10	**Reserved**	Project:	DevCTG	Format:	MBZ
9:0	**Delay**	Project:	DevCTG	Format:	U10
	Minimum number of micro-seconds allowed between preemptions.				

8.15.7 CLKCMP — Compare count clock stop (Debug)

CLKCMP — Compare count clock stop (Debug)

Register Type:	MMIO
Address Offset:	2360h
Project:	All
Default Value:	0000 0000 0000 0000h
Access:	R/W This register is *not* set by the context restore.
Size (in bits):	64

This register stores the value of the count of clock ticks that should cause the clock to stop. An internal hardware counter keeps track of the clock ticks. The internal hardware counter is reset when this register is written.

The reference clock used by this counter is the core render clock (crclk). **Crclk** is chosen here specifically because it is the operating frequency for a majority of the logic in the 3D pipeline. See the EDS for details for the frequency of the crclk. See section 1.21.

Bit	Description
63:0	**Clock Stop Value** | Project: All | Format: U64 This register reflects the total number of crclk ticks that need to pass before the crclk is stopped. A write to this register causes the internal clock counter to reset.

8.15.8 VFDC—Set Value of Draw Count (DEBUG)

VFDC—Set Value of Draw Count (DEBUG)

Register Type:	MMIO
Address Offset:	2450h
Project:	All
Default Value:	UUUU UUUUh
Access:	R/W
Size (in bits):	32

The VFDC register is to set the initial DRAW count starting point. This is needed to be able to reset and start at different draw counts.

Bit	Description
31:24	**Reserved** Project: All Format: MBZ
23:0	**Set Value of Draw Count** Project: All Format: U24 This value must be set before enabling the **Skip Initial Primitive or Max Primitives Limit Enable**. If not then the start of the Draw Count is undefined.

8.15.9 VFSKPD—VF Scratch Pad (DEBUG)

VFSKPD—VF Scratch Pad (DEBUG)

Register Type:	MMIO
Address Offset:	2470h
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

Bit	Description
31:16	**Mask Bits** Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All bits implemented)
15	**SnapShot Continue** Project: All Format: U1 Write a ‘1’ to this field with the mask will allow VF to continue once a SnapShot occurs. Writing a ‘0’ has no effect.
14:3	**Reserved** Project: All Format: MBZ

VFSKPD—VF Scratch Pad (DEBUG)

2

Vertex Cache Implicit Disable Inhibit	
Project:	All
Default Value:	0h
Format:	U1

Value	Name	Description	Project
0h		Allow VF to disable VS0 when Sequential index or Prim ID is a valid Element.	All
1h		VF never implicitly disables the vertex cache. Software must disable the VS0 Cache when required.	All

1

Disable Over Fetch Cache	
Project:	All
Default Value:	0h
Format:	Disable

Value	Name	Description	Project
0h		Cache will check for data in cache before making a request to memory	All
1h		Always re-fetch new data from memory.	All

0

Disable Pending FIFO	
Project:	All
Default Value:	0h
Format:	Disable

Value	Name	Description	Project
0h		Allow VFunit to request TLB data without waiting for pending TLB data to return.	All
1h		Only allow one pending TLB request at a time	All

8.16 Pipelines Statistics Counter Registers

These registers keep continuous count of statistics regarding the 3D pipeline. They are saved and restored with context but should not be changed by software except to reset them to 0 at context creation time. These registers may be read at any time; however, to obtain a meaningful result, a pipeline flush just prior to reading the registers is necessary in order to synchronize the counts with the primitive stream.

8.16.1 IA_VERTICES_COUNT — Reported Vertices Counter

IA_PRIMITIVES_COUNT	
Register Type:	MMIO
Address Offset:	2310h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	64
Trusted Type:	1
This register stores the count of vertices processed by VF. This register is part of the context save and restore.	

Bit	Description
63:0	**IA Vertices Count Report** Total number of vertices fetched by the VF stage. This count is updated for every input vertex as long as **Statistics Enable** is set in VF_STATE (see the Vertex Fetch Chapter in the *3D* Volume.)

8.16.2 IA_PRIMITIVES_COUNT — Reported Vertex Fetch Output Primitives Counter

IA_PRIMITIVES_COUNT	
Register Type:	MMIO
Address Offset:	2318h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	64
Trusted Type:	1
This register stores the count of primitives generated by VF. This register is part of the context save and restore.	

Bit	Description
63:0	**IA Primitives Count Report** Total number of primitives output by the Vertex Fetch (IA) stage. This count is updated for every primitive *output* by the VF stage, as long as **Statistics Enable** is set in VF_STATE (see the Vertex Fetch Chapter in the *3D* Volume.)

8.16.3 VS_INVOCATION_COUNT— Reported Vertex Shader Invocation Counter

VS_INVOCATION_COUNT	
Register Type:	MMIO
Address Offset:	2320h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	64
Trusted Type:	1
This register stores the value of the vertex count shaded by VS. This register is part of the context save and restore	

Bit	Description
63:0	**VS Invocation Count Report** Number of vertex shader threads invoked by the VS stage. Updated only when **Statistics Enable** is set in VS_STATE (see the Vertex Shader Chapter in the *3D* Volume.)

8.16.4 GS_INVOCATION_COUNT — Reported Geometry Shader Thread Invocation Counter

GS_INVOCATION_COUNT	
Register Type:	MMIO
Address Offset:	2328h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	64
Trusted Type:	1
This register stores the number of invoked geometry shader threads. This register is part of the context save and restore.	

Bit	Description
63:0	**GS Invocation Count** Number of geometry shader threads invoked by the GS stage. Updated only when **Statistics Enable** is set in GS_STATE (see the Geometry Shader Chapter in the *3D* Volume.)

8.16.5 GS_PRIMITIVES_COUNT — Reported Geometry Shader Output Primitives Counter

GS_PRIMITIVES_COUNT

Register Type:	MMIO
Address Offset:	2330h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	64
Trusted Type:	1

This register reflects the total number of primitives that have been output by the Geometry Shader stage. This register is part of the context save and restore.

Bit	Description
63:0	**GS Primitives Count** Total number of primitives output by the geometry stage. This count is updated for every primitive *output* by the geometry stage, as long as **GS Output Object Statistic Enable** is set in CLIP_STATE (see the Clipper and Geometry Shader Chapters in the *3D* Volume.)

8.16.6 CL_INVOCATION_COUNT— Reported Clipper Thread Invocation Counter

CL_INVOCATION_COUNT

Register Type:	MMIO
Address Offset:	2338h
Project:	Pre-DevIL
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	64
Trusted Type:	1

This register stores the count of invoked clipper threads. This register is part of the context save and restore.

Bit	Description
63:0	**CL Invocation Count Report** Number of clipper threads invoked by the clipper stage. Updated only when **Statistics Enable** is set in CLIP_STATE (see the Clipper Chapter in the *3D* Volume.)

CL_INVOCATION_COUNT

Register Type:	MMIO
Address Offset:	2338h
Project:	DevIL+
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	64
Trusted Type:	1

This register stores the count of objects entering the Clipper stage. This register is part of the context save and restore.

Bit	Description
63:0	**CL Invocation Count Report** Number of objects entering the clipper stage. Updated only when **Statistics Enable** is set in CLIP_STATE (see the Clipper Chapter in the *3D* Volume.)

8.16.7 CL_PRIMITIVES_COUNT— Reported Clipper Output Primitives Counter

CL_PRIMITIVES_COUNT

Register Type:	MMIO
Address Offset:	2340h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	64
Trusted Type:	1

This register reflects the total number of primitives that have been output by the clipper. This register is part of the context save and restore.

Bit	Description
63:0	**Clipped Primitives Output Count** Total number of primitives output by the clipper stage. This count is updated for every primitive *output* by the clipper stage, as long as **Statistics Enable** is set in SF_STATE (see the Clipper and SF Chapters in the *3D* Volume.)

8.16.8 PS_INVOCATION_COUNT— Reported Pixels Shaded counter

PS_INVOCATION_COUNT

Register Type:	MMIO
Address Offset:	2348h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	64
Trusted Type:	1

This register stores the value of the count of pixels that get shaded. This register is part of the context save and restore.

Bit	Description
63:0	**PS Invocation Count** Reflects a count of the total number of pixels that are dispatched to pixel shader invocations while **Statistics Enable** is set in the Windower. See the Windower chapter of the *3D* volume for details. This count will generally be much greater than the actual count of PS threads since a single thread may process up to 32 pixels.

8.16.9 PS_DEPTH_COUNT — Reported Pixels Passing Depth Test Counter

PS_DEPTH_COUNT

Register Type:	MMIO
Address Offset:	2350h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	64
Trusted Type:	1

This register stores the value of the count of pixels that have passed the depth test. This register is part of the context save and restore. Note that the value of this register can be obtained in a pipeline-synchronous fashion without a pipeline flush by using the 3DCONTROL command. See 3D Overview in the 3D volume.

Bit	Description
63:0	**Depth Count** This register reflects the total number of pixels that have passed the depth test (i.e., will be visible). All pixels are counted when **Statistics Enable** is set in the Windower State. See the Windower chapter of the *3D* volume for details. Pixels that pass the depth test but fail the stencil test will *not* be counted.

8.16.10 TIMESTAMP — Reported Timestamp Count

TIMESTAMP — Reported Timestamp Count

Register Type:	MMIO
Address Offset:	2358h
Project:	All
Default Value:	0000 0000 0000 0000h
Access:	R/W. This register is *not* set by the context restore.
Size (in bits):	64

This register stores the value of the count of clock ticks that have passed since it was last reset. Note that the value of this register can be obtained in a 3D pipeline-synchronous fashion without a pipeline flush by using the 3DCONTROL command. See 3D Pipeline in the *3D and Media* volume.

The reference clock used by this counter is the GMCH core and Processor-Side Bus (PSB) clock referred to as "**hclk**". **Hclk** is not used elsewhere in the graphics device and is chosen here specifically because it is not subject to throttling as the graphics device clock is. The **hclk** used is not gated, throttled or selectively powered down so that the TIMESTAMP can remain accurate even during power management activity (as long as the GMCH does not have all of its clocks stopped, as when it is fully powered down.)

The frequency of **hclk** is determined externally to the GMCH and can be discovered through the "Clocking Configuration" ("CLKCFG") MCHBAR register. See the EDS for details. Note that the MCHBAR registers can be accessed through the MCHBAR aperture in MMIO space. See section 8.23.

TIMESTAMP is *not* reset by a graphics reset. It will maintain its value unless a full chipset reset is performed.

<table>
<tr><th>Bit</th><th colspan="5">Description</th></tr>
<tr><td rowspan="2">63:0</td><td><b>TIMESTAMP</b></td><td>Project:</td><td>All</td><td>Format:</td><td>U64</td></tr>
<tr><td colspan="5">This register reflects the total number of ticks that have passed since reset or the last time 0000 0000 0000 0000h was written to this register. SW should not write a non-zero value to this register. The value in this register increments once every 16 hclks. A full GMCH reset is required to reset this register; since this register is in the <b>hclk</b> domain it is not reset by a graphics reset alone.</td></tr>
</table>

8.16.11 TIMESTAMP — Reported Timestamp Count ([DevCTG] Only)

TIMESTAMP — Reported Timestamp Count

Register Type:	MMIO
Address Offset:	2358h
Project:	DevCTG
Default Value:	0000 0000 0000 0000h
Access:	R/W. This register is *not* set by the context restore.
Size (in bits):	64

This register provides an elapsed real-time value that can be used as a timestamp for GPU events over short periods of time. Note that the value of this register can be obtained in a 3D pipeline-synchronous fashion without a pipeline flush by using the PIPE_CONTROL command. See *3D Geometry Pipeline* in the "3D and Media" volume.

This register (effectively) counts at a constant frequency by adjusting the increment amount according to the actual reference clock frequency. SW therefore does not need to know the reference clock frequency.

This register is *not* reset by a graphics reset. It will maintain its value unless a full chipset reset is performed. This register will be reset to 0 if either DW is written to (any value).

Bit	Description				
63:32	**TIMESTAMP**	Project:	All	Format:	U32
	This register represents 1.024 uS of time.				
31:20	**TIMESTAMP**	Project:	All	Format:	U12
	This register represents ¼ nS increment of the time stamp.				
19:0	**Reserved**	Project:	All	Format:	MBZ

8.16.12 VT_CL_WRITTEN— Number of Cachelines for the GTT used for VT-d purposes (Debug/Validation Only)([DevCTG] only)

CL_WRITTEN— Number of Cachelines for the GTT used for VT-d purposes (Debug/Validation Only)

Register Type:	MMIO
Address Offset:	2370h
Project:	DevCTG
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

This register is used in conjunction with a chipset debug register bit (MCHBAR offset 44 bit10) to limit the number of cachelines the graphics uses for the size of the GTT.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:2</td><td><table>
<tr><td><b>Cacheline Count</b></td><td>Project:</td><td>DevCTG</td><td>Format:</td><td>U30</td></tr>
<tr><td colspan="5">Total number of cachelines to use for the size of the GTT</td></tr>
</table></td></tr>
<tr><td>1</td><td><table>
<tr><td colspan="3"><b>Enable</b></td></tr>
<tr><td>Project:</td><td colspan="2">DevCTG</td></tr>
<tr><td>Default Value:</td><td>0h</td><td></td></tr>
<tr><td>Format:</td><td>Enable</td><td></td></tr>
<tr><td colspan="3"></td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>Disable</td><td>Do not use the cacheline value in this register for the GTT size</td><td>DevCTG</td></tr>
<tr><td>1h</td><td>Enable</td><td>Use current register to indicate the number of cachelines for GTT size</td><td>DevCTG</td></tr>
</table></td></tr>
<tr><td>0</td><td><table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>DevCTG</td><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
</table>

8.16.13 SO_NUM_PRIMS_WRITTEN— Reported Stream Output Num Primitives Written Counter ([DevCTG] Only)

<table>
<tr><td colspan="2">SO_NUM_PRIMS_WRITTEN— Reported Stream Output Num Primitives Written Counter</td></tr>
<tr><td><b>Register Type:</b></td><td>MMIO</td></tr>
<tr><td><b>Address Offset:</b></td><td>2288h</td></tr>
<tr><td><b>Project:</b></td><td>DevCTG+</td></tr>
<tr><td><b>Default Value:</b></td><td>0000 0000 0000 0000h</td></tr>
<tr><td><b>Access:</b></td><td>R only. This register is set by the context restore.</td></tr>
<tr><td><b>Size (in bits):</b></td><td>64</td></tr>
<tr><td colspan="2">This register is used to (indirectly) count the number of primitives which GS threads have successfully written to Streamed Vertex Output buffers. This register is part of the context save and restore.
<b>[Errata]</b> This regiser gets reset when write happens to register 2380h</td></tr>
<tr><td><b>Bit</b></td><td><b>Description</b></td></tr>
<tr><td>63:0</td><td><table><tr><td><b>Num Prims Written Count</b></td><td>Project:</td><td>DevCTG</td><td>Format:</td><td>U64</td></tr><tr><td colspan="5">This count is incremented (by one) every time a GS thread outputs a DataPort Streamed Vertex Buffer Write message with the <b>Increment Num Prims Written</b> bit set in the message header (see the <i>Geometry Shader</i> and <i>Data Port</i> chapters in the <i>3D</i> Volume.)</td></tr></table></td></tr>
</table>

8.16.14 SO_PRIM_STORAGE_NEEDED — Reported Stream Output Primitive Storage Needed Counter ([DevCTG] Only)

<table>
<tr><td colspan="2">SO_PRIM_STORAGE_NEEDED — Reported Stream Output Primitive Storage Needed Counter</td></tr>
<tr><td><b>Register Type:</b></td><td>MMIO</td></tr>
<tr><td><b>Address Offset:</b></td><td>2280h</td></tr>
<tr><td><b>Project:</b></td><td>DevCTG+</td></tr>
<tr><td><b>Default Value:</b></td><td>0000 0000 0000 0000h</td></tr>
<tr><td><b>Access:</b></td><td>R only. This register is set by the context restore.</td></tr>
<tr><td><b>Size (in bits):</b></td><td>64</td></tr>
<tr><td colspan="2">This register is used to (indirectly) count the number of primitives which GS threads would have written to Streamed Vertex Output buffers if all buffers had been large enough to accommodate the writes . This register is part of the context save and restore.
<b>[Errata]</b> This register gets reset when write happens to register 2388h</td></tr>
<tr><td><b>Bit</b></td><td><b>Description</b></td></tr>
<tr><td>63:0</td><td><table><tr><td><b>Prim Storage Needed Count</b></td><td>Project:</td><td>DevCTG</td><td>Format:</td><td>U64</td></tr><tr><td colspan="5">This count is incremented (by one) every time a GS thread outputs a DataPort Streamed Vertex Buffer Write message with the <b>Increment Prim Storage Needed</b> bit set in the message header (see the <i>Geometry Shader</i> and <i>Data Port</i> chapters in the <i>3D</i> Volume.)</td></tr></table></td></tr>
</table>

8.17 MTCH_CID_RST – Matched Context ID Reset Register

MTCH_CID_RST – Matched Context ID Reset Register

Register Type:	MMIO
Address Offset:	2524h
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

This register is used to generate a Context ID specific reset (Render Only). To initiate a reset, the register is written with the pending bit set. Hardware compares the current context ID with the register and on match generates a Render Only reset. After reset is complete, HW clears the pending bit and can be programmed to generate an interrupt. The match bit is set. If the current context ID does not match this register, the pending bit is reset and an interrupt is generated. The match bit is reset.

The match indicates the result of the last comparison, and its valid only when pending bit is zero.

Please see MCIDRST interrupt bit assignment in the Interrupt Control Registers.

Bit	Description				
31:12	**Reserved**	Project:	All	Format:	MBZ
11:2	**Reserved**	Project:	All	Format:	MBZ
1	**Reserved**	Project:	All	Format:	MBZ
0	**Reserved**	Project:	All	Format:	MBZ

8.18 Display Related Registers for Flip Queue

8.18.1 MAXQ_FLIP_A – Maximum Flips Allowed for Display A Register ([DevCTG] Only)

MAXQ_FLIP_A – Maximum Flips Allowed for Display A Register

Register Type:	MMIO
Address Offset:	2530h
Project:	DevCTG
Default Value:	0000 0004h
Access:	R/W
Size (in bits):	32

This register is *not* saved or restored with context. The render-only reset will not affect this register.

Bit	Description
31:5	**Reserved** Project: DevCTG Format: MBZ
4:0	**Maximum Flips allowed** Project: DevCTG Format: U5 The number of flip buffers allocated by the software to support queued flips for Display plane A. For a double buffered case, the value programmed into this register should be 1. [DevCTG]: Maximum value in this register is 8

8.18.2 MAXQ_FLIP_B – Maximum Flips Allowed for Display B Register ([DevCTG] Only)

MAXQ_FLIP_B – Maximum Flips Allowed for Display B Register

Register Type:	MMIO
Address Offset:	2534h
Project:	DevCTG
Default Value:	0000 0004h
Access:	R/W
Size (in bits):	32

This register is *not* saved or restored with context. The render-only reset will not affect this register.

Bit	Description
31:5	**Reserved** Project: DevCTG Format: MBZ
4:0	**Maximum Flips allowed** Project: DevCTG Format: U5 The number of flip buffers allocated by the software to support queued flips for Display plane B. For a double buffered case, the value programmed into this register should be 1. [DevCTG]: Maximum value in this register is 8

8.18.3 NUM_FLIP_A – Number of flips pending on Display A Register ([DevCTG] Only)

NUM_FLIP_A – Number of flips pending on Display A Register

Register Type:	MMIO
Address Offset:	2538h
Project:	DevCTG
Default Value:	0000 0000h
Access:	R/W [Debug only]
Size (in bits):	32

This register is *not* saved or restored with context. The render-only reset will not affect this register.

Bit	Description
31:5	**Reserved** Project: DevCTG Format: MBZ
4:0	**Number of Flips pending** Project: DevCTG Format: U5 The number of flips pending in the hardware for Display plane A.

8.18.4 NUM_FLIP_B – Number of flips pending on Display B Register ([DevCTG] Only)

NUM_FLIP_B – Number of flips pending on Display B Register

Register Type:	MMIO
Address Offset:	253Ch
Project:	DevCTG
Default Value:	0000 0000h
Access:	R/W [Debug only]
Size (in bits):	32

This register is *not* saved or restored with context. The render-only reset will not affect this register.

Bit	Description
31:5	**Reserved** Project: DevCTG Format: MBZ
4:0	**Number of Flips pending** Project: DevCTG Format: U5 The number of flips pending in the hardware for Display plane B.

8.19 Video Codec Engine Command Streamer

[DevBW] and [DevCL] do not support the Video Codec Engine (VCE).

For [DevCTG, VCE is also referred to as the Bit-Serial Decoder (BSD) Engine, as only the front portion of the multi-format decoder is implemented.

VCE has its own command streamer and operates completely independently of the render (3D/Media) pipeline command streamer.

This command streamer supports a completely independent set of registers. Only a subset of the MI Registers is supported for this 2nd command streamer. The effort is to keep the registers at the same offset as the render command streamer registers. The base of the registers for the video decode engine will be defined per project, the offsets will be maintained.

Project	Base Adress Value for the memory interface register offset for the Bit Stream Command Stream
DevCTG	0x2000 Eg: The Ring buffer tail pointer will be 0x2000 + 0x2030

8.19.1 Registers in the VCE Command Streamer [DevCTG+]

Each register is at the same offset from 04000h as its primary counterpart is offset from 02000h.

8.19.1.1 BCS_EXCC—Execute Condition Code Register

EXCC—Execute Condition Code Register	
Register Type:	MMIO
Address Offset:	2028h
Project:	All
Default Value:	00000000h
Access:	R/W,RO
Size (in bits):	32
Trusted Type:	1

This register contains user defined and hardware generated conditions that are used by MI_WAIT_FOR_EVENT commands. An MI_WAIT_FOR_EVENT instruction excludes the executing ring from arbitration if the selected event evaluates to a “1”, while instruction is discarded if the condition evaluates to a “0”. Once excluded a ring is enabled into arbitration when the selected condition evaluates to a “0”.

Bit	Description
31:18	**Reserved** Project: All Format: MBZ
17	**Mask Bits** Format: Mask[1] This bit serves as a write enable for bit 1. If this register is written with this bit clear the corresponding bit in the field 1 will not be modified. Reading these bits always returns 0s.
16	**Mask Bits** Format: Mask[0] These bits serves as a write enable for bit 0. If this register is written with any of these bits clear the corresponding bit in the field 0 will not be modified. Reading these bits always returns 0s.
15:7	**Reserved** Project: All Format: MBZ
6	**Reserved**
5:2	**Reserved** Project: All Format: MBZ
1	**Blitter Command Streamer Condition Codes** The software may signal a Stream Semaphore by setting the Mask bit and Signal Bit together to match the bit field specified in a WAIT_FOR_EVENT (Semaphore).
0	**Render Command Streamer Condition Codes** The software may signal a Stream Semaphore by setting the Mask bit and Signal Bit together to match the bit field specified in a WAIT_FOR_EVENT (Semaphore).

8.19.1.2 BCS_RINGBUF—Ring Buffer Registers

Address Offset:	04030h – 0403Fh: Ring Buffer: offset 0h = _TAIL offset 4h = _HEAD offset 8h = _START offset Ch = _CTL
Default Value:	0000 0000h
Access:	Read/32 bit Write Only
Size:	4 DWords / Ring Buffer

These registers are used to define and operate the "ring buffer" mechanism which can be used to pass instructions to the command interface. The buffer itself is located in a linear memory region. The ring buffer is defined by a 4 Dword register set that includes starting address, length, head offset, tail offset, and control information. Refer to the *Programming Interface* chapter for a detailed description of the parameters specified in this ring buffer register set, restrictions on the placement of ring buffer memory, arbitration rules, and in how the ring buffer can be used to pass instructions.

Ring Buffer Head and Tail Offsets must be properly programmed before it is enabled. A Ring Buffer can be enabled when empty.

The format of the Ring Buffer register set follows:

DWord Offset	Bit	Description
0	31:21	Reserved: MBZ
	20:3	**Tail Offset:** This field is written by software to specify where the valid instructions placed in the ring buffer end. The value written points to the QWord *past* the last valid QWord of instructions. In other words, it can be defined as the *next* QWord that software will write instructions into. Software must write subsequent instructions to QWords following the Tail Offset, possibly wrapping around to the top of the buffer (i.e., software can't skip around within the buffer). Note that all DWords prior to the location indicated by the **Tail Offset** must contain valid instruction data – which may require instruction padding by software. See **Head Offset** for more information. Format = U18 QWord Offset
	2:0	Reserved: MBZ
1	31:21	**Wrap Count:** This field is incremented by 1 whenever the **Head Offset** wraps from the end of the buffer back to the start (i.e., whenever it wraps back to 0). Appending this field to the **Head Offset** field effectively creates a virtual 4GB Head "Pointer" which can be used as a tag associated with instructions placed in a ring buffer. The Wrap Count itself will wrap to 0 upon overflow. The Wrap Count will get cleared as a result of writes of the Starting Address field. Format = U11 count of ring buffer wraps

DWord Offset	Bit	Description
	20:2	**Head Offset:** This field indicates the offset of the *next* instruction DWord to be parsed. Software will initialize this field to select the first DWord to be parsed once the RB is enabled. (Writing the Head Offset while the RB is enabled is UNDEFINED). Subsequently, the device will increment this offset as it executes instructions – until it reaches the QWord specified by the **Tail Offset**. At this point the ring buffer is considered "empty". **Programming Notes:** • A RB can be enabled empty or containing some number of valid instructions. • Head Offset is cleared as a result of writes of the Starting Address field. Format = U19 DWord Offset
	1:0	Reserved: MBZ
2	31:12	**Starting Address:** This field specifies Bits 31:12 of the 4KB-aligned starting Graphics Address of the ring buffer. Writing this register also causes the Head Offset to be reset to zero, and the Wrap Count to be reset to zero. All ring buffer pages must map to Main Memory (uncached) pages. Ring Buffer addresses are always translated through the global GTT. Per-process address space can only be used via a batch buffer with the appropriate **Memory Space Select**. If the run list mechanism (register RNCID) is used to submit a context to run, this field will be set to a page (4K) offset from LRCA. Format: Graphics Address Bits 31:12
	11:0	Reserved: MBZ
3	31:21	Reserved: MBZ

	20:12	**Buffer Length:** This field is written by SW to specify the length of the ring buffer in 4 KB Pages.
		If the run list mechanism (register RNCID) is used to submit a context to run, this field will be set to 4 pages (16K).
		Format = U9 in 4 KB pages – 1
		Range = [0 = 1 page = 4 KB, 1FFh = 512 pages = 2 MB]

	11:9	Reserved: MBZ
	8	**Disable Register Accesses:** 0 = Ring is allowed to access (read or write) MMIO space. 1 = Ring is not allowed to write MMIO space. Ring *is* allowed to read registers.

DWord Offset	Bit	Description
	7:3	Reserved: MBZ
	2:1	**Automatic Report Head Pointer:** This field is written by software to control the automatic “reporting” (write) of this ring buffer’s “Head Pointer” register (register DWord 1) to the corresponding location within the Hardware Status Page. Automatic reporting can either be disabled or enabled at 4KB, 64KB or 128KB boundaries within the ring buffer. Format = 0: MI_AUTOREPORT_OFF – Automatic reporting disabled 1: MI_AUTOREPORT_64KB – Report every 16 pages (64KB) 2: MI_AUTOREPORT_4KB – Report every page (4KB) [DevCTG] Only 3: MI_AUTOREPORT_128KB – Report every 32 pages (128KB) **[DevCTG]** When the **Per-Process Virtual Address Space and Run List Enable** bit is set and automatic head reporting is desired, this field must be set to option 2 since the ring buffer will be only 16KB in size. The head pointer will be reported to the head pointer location in the PP HW Status Page when it passes each 4KB page boundary. When the above-mentioned bit is set, reporting will behave just as on the prior devices (as documented above), and option 2 is not legal.
	0	**Ring Buffer Enable:** This field is used to enable or disable this ring buffer. It can be enabled or disabled regardless of whether there are valid instructions pending. Format = Enable

8.19.1.3 BCS_HWS_PGA — Hardware Status Page Address Register

Address Offset:	04080h–04083h
Default Value:	1FFF F000h
Access:	Read/Write
Size:	32 bits

This register is used to program the 4 KB-aligned System Memory address of the Hardware Status Page used to report hardware status into (typically cacheable) System Memory.

Bit	Description
31:12	**Address:** This field is used by SW to specify Bits 31:12 of the 4 KB-aligned System Memory address of the 4 KB page known as the “Hardware Status Page”. Bits 11:0 of the address MBZ. Format = Bits 31:12 of Graphics Memory Address
11:0	Reserved: MBZ

The following table defines the layout of the Hardware Status Page:

DWord Offset	Description
3:0	**Reserved.** Must not be used.
4	**Head Pointer Storage:** The contents of the Ring Buffer Head Pointer register (register DWord 1) are written to this location either as result of an MI_REPORT_HEAD instruction or as the result of an “automatic report” (see RINGBUF registers).
0Fh:05h	**Reserved.** Must not be used.
(3FFh – 010h)	These locations can be used for general purpose via the MI_STORE_DATA_INDEX or MI_STORE_DATA_IMM instructions.

8.19.1.4 BCS_NOPID — NOP Identification Register

Address Offset:	04094h–04097h
Default Value:	0000 0000h
Access:	Read Only
Size:	32 bits

The BCS_NOPID register contains the Noop Identification value specified by the last MI_NOOP instruction that enabled this register to be updated.

Bit	Description
31:22	Reserved: MBZ
21:0	**Identification Number:** This field contains the 22-bit Noop Identification value specified by the last MI_NOOP instruction that enabled this field to be updated.

8.19.1.5 BCS_MI_MODE — Mode Register for Software Interface

Address Offset:	0409Ch–0409Fh
Default Value:	0000 0000h
Access:	Read/Write
Size:	32 bits

The MI_MODE register contains information that controls software interface aspects of the command parser.

Bit	Description
31:16	**Masks:** A “1” in a bit in this field allows the modification of the corresponding bit in Bits 15:0
15:12	**Reserved** Read/Write
11	**Invalidate UHPTR enable:** If bit set H/W clears the valid bit of BCS_UHPTR (4134h, bit 0) when current active head pointer is equal to UHPTR.
10	**Reserved** Read/Write
9	**Ring Idle (Read Only Status bit)** 0 = Parser not Idle 1 = Parser Idle ***Writes to this bit are not allowed.***

Bit	Description
8	**Stop Ring** 0 = Normal Operation. 1 = Parser is turned off. Software must set this bit to force the Ring and Command Parser to Idle. Software must read a "1" in Ring Idle bit after setting this bit to ensure that the hardware is idle. ***Software must clear this bit for Ring to resume normal operation.***
7:2	**Reserved** Read/Write
1	**Dummy Read Disable.** Nominally a command stream flush is completed with a dummy read to memory to push all pending writes. Setting this bit to a "1" disables the dummy read.
0	**Reserved** Read/Write

8.19.1.6 BCS_INSTPM—Instruction Parser Mode Register

Address Offset:	040C0h–040C3h
Default Value:	0000 0000h
Access:	Read/Write
Size:	32 bits

The BCS_INSTPM register is used to control the operation of the BCS Instruction Parser. Certain classes of instructions can be disabled (ignored) – often useful for detecting performance bottlenecks. Also, "Synchronizing Flush" operations can be initiated – useful for ensuring the completion (vs. only parsing) of rendering instructions.

Programming Notes:

- All Reserved bits are implemented.

Bit	Description
31:16	**Masks:** These bits serve as write enables for bits 15:0. If this register is written with any of these bits clear the corresponding bit in the field 15:0 will not be modified. Reading these bits always returns 0s.
15:6	Reserved: MBZ
5	**Sync Flush Enable:** This field is used to request a Sync Flush operation. The device will automatically clear this bit before completing the operation. See Sync Flush (*Programming Environment*). **Programming Note:** • The command parser must be stopped prior to issuing this command by setting the **Stop Ring** bit in register **BCS_MI_MODE**. Only after observing **Ring Idle** set in **BCS_MI_MODE** can a Sync Flush be issued by setting this bit. Once this bit becomes clear again, indicating flush complete, the command parser is re-enabled by clearing **Stop Ring**. Format = Enable (cleared by HW)
4:0	Reserved: MBZ

8.19.1.7 BCS_UHPTR — Pending Head Pointer Register

Address Offset:	04134h–04137h
Default Value:	0000 0000h
Access:	Read/Write
Size:	32 bits

Bit	Description
31:3	**Head Pointer Address**: This register represents the GFX address offset where execution should continue in the ring buffer following execution of an MI_ARB_CHECK command. Format = MI_Graphics_Offset
2:1	Reserved: MBZ
0	**Head Pointer Valid:** 1 = Indicates that there is an updated head pointer programmed in this register 0 = No valid updated head pointer register, resume execution at the current location in the ring buffer This bit is set by the software to request a pre-emption. It is reset by hardware after the head pointer in this register is read. The hardware uses the head pointer programmed in this register at the time the reset is generated.

8.19.1.8 BCS_CNTR—Counter for the bit stream decode engine

Address Offset:	04178h–0417Bh
Default Value:	FFFF FFFFh
Access:	Read/Write
Size:	32 bits

Bit	Description
31:0	**Count Value**: Writing a Zero value to this register starts the counting. Writing a Value of FFFF FFFF to this counter stops the counter

8.19.1.9 BCS_THRSH—Threshold for the counter of bit stream decode engine

Address Offset:	0417Ch–0417Fh
Default Value:	00014500h
Access:	Read/Write
Size:	32 bits

Bit	Description
31:0	**Threshold Value**: The value in this register reflects the number of clocks the bit stream decode engine is expected to run. If the value is exceeded the counter is reset and an interrupt may be enabled in the device.

8.19.1.10 BCS_BB_ADDR—Batch Buffer Head Pointer Register

Address Offset:	02140h–02147h
Default Value:	0000 0000 0000 0000h
Access:	Read-Only
Size:	64 bits

This register contains the current QWord Graphics Memory Address of the last-initiated batch buffer.

Bit	Description
63:32	Reserved: MBZ
31:3	**Batch Buffer Head Pointer:** This field specifies the QWord-aligned Graphics Memory Address where the last initiated Batch Buffer is currently fetching commands. If no batch buffer is currently active, the Valid bit will be 0 and this field will be meaningless. .
2:1	Reserved: MBZ
0	**Valid:** 1 = Batch buffer Valid 0 = Batch buffer Invalid

8.19.1.11 BCS_ECOSKPD—ECO Scratch Pad (DEBUG)

Address Offset:	041D0h–041D3h
Default Value:	0000 0000h
Access:	Read/Write
Size:	32 bits

Bit	Description
31:16	**Mask Bits**: Must be set to modify corresponding bit in Bits 15:0. (All bits implemented)
15:12	**Reserved:** MBZ
11	**Must be set to 1**
10:0	**Reserved:** MBZ

8.19.1.12 Two-Level Per-Process Virtual Memory ([DevCTG] Only)

[DevCTG] Supports a 2-level mapping scheme for PPGTT, consisting of a first-level page directory containing page table base addresses, and the page tables themselves on the 2nd level, consisting of page addresses.

Two-Level Per-Process Virtual Memory ([DevCTG] Only).

8.19.1.12.1 BCS_PP_DIR_BASE – Page Directory Base Register

<table>
<tr><td colspan="2"><b>BCS_PP_DIR_BASE – Page Directory Base Register</b></td></tr>
<tr><td><b>Register Type:</b></td><td>MMIO</td></tr>
<tr><td><b>Address Offset:</b></td><td>4390h</td></tr>
<tr><td><b>Project:</b></td><td>All</td></tr>
<tr><td><b>Default Value:</b></td><td>00000000h</td></tr>
<tr><td><b>Access:</b></td><td>R/W</td></tr>
<tr><td><b>Size (in bits):</b></td><td>32</td></tr>
<tr><td><b>Trusted Type:</b></td><td>1</td></tr>
<tr><td colspan="2">This register contains the offset into the GGTT where the (current context's) PPGTT page directory begins. This register is restored with context</td></tr>
</table>

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:16</td><td>
<table>
<tr><td colspan="3"><b>Page Directory Base Offset</b></td></tr>
<tr><td>Project:</td><td colspan="2">DevCTG+</td></tr>
<tr><td>Security:</td><td colspan="2">None</td></tr>
<tr><td>Default Value:</td><td>0h</td><td>DefaultVaueDesc</td></tr>
<tr><td>Format:</td><td>U15</td><td></td></tr>
<tr><td>Address:</td><td colspan="2">GraphicsAddress[31:16]</td></tr>
<tr><td>Range</td><td colspan="2">[0,PPGTT Size - 1 in cachelines]</td></tr>
<tr><td colspan="3">Contains the cacheline (64-byte) address into the GGTT where the page directory begins.</td></tr>
</table>
</td></tr>
<tr><td>15:0</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
</table>

8.19.1.12.2 BCS_PP_DCLV – PPGTT Directory Cacheline Valid Register

BCS_PP_DCLV – PPGTT Directory Cacheline Valid Register

Register Type:	MMIO
Address Offset:	4398h
Project:	All
Default Value:	00000000h; 00000000h; 00000000h; 00000000h
Access:	RO
Size (in bits):	2x32
Trusted Type:	1

This register can be used to designate entire cachelines of the PPGTT Directory as invalid. Bits that are set indicate the corresponding 16 directory entry group is valid. Note that some or all of the entries could have their valid bits clear, indicating they are invalid. This register is restored with context (prior to restoring the on-chip directory cache itself). This register is also restored when switching to a context whose LRCA matches the current CCID if the **Force PD Restore** bit is set in the context descriptor.

The context image of this register must be updated and maintained by SW.

This register can effectively be used to limit the size of a processes' virtual address space. Any access by a process that requires a PD entry in a set that is not enabled in this register will cause a fatal error, and no attempt to fetch the PD entry will be made.

DWord	Bit	Description
0	31:0	PPGTT Directory Cacheline Valid. Project: All Security: None Default Value: 0h DefaultVaueDesc Format: U32 FormatDesc If set, each bit in this register corresponds to 16 valid entries of the page directory. If clear, these entries are considered invalid and fetch of these entries will not be attempted.
1	31:0	**Reserved** Project: All Format: MBZ

8.19.2 Unique BCS Registers ([DevCTG] Only)

The Video Decode Command Streamer supports a simplified run list mechanism for the purpose of submitting command buffers to it for execution. Two run lists are supported, each with a single context element. One context can be running and one pending.

8.19.2.1 BCS_RCCID—Ring Buffer Current Context ID Register

Address Offset:	04190h–04193h
Default Value:	00 00 00 00h
Access:	Read/Write
Size:	32 bits

This register contains the current "ring context ID" associated with the ring buffer.

Programming Notes:

- The current context registers must not be written directly (via MMIO). The RCCID register should only be updated indirectly from RNCID.

Bit	Description

31:12	**Logical Ring Context Address (LRCA):**
	This field contains the 4 KB-aligned Memory address of the current Ring Context Descriptor associated with this ring buffer. See the RNCID register for the Descriptor format.
	Format = GlobalGraphicsVirtualAddress[31:12]

11:1	Reserved: MBZ
0	**Valid:** 1 = The other field of this register is valid. A ring context is executing and the LRCA field contains the address of its context descriptor. 0 = The other field of this register is invalid. No ring context is executing. This streamer is idle or it is being used in Basic Scheduler mode where the ring buffer registers are manipulated directly and no ring context is used.

8.19.2.2 BCS_RNCID—Ring Buffer Next Context ID Register

Address Offset:	04194h–04197h
Default Value:	00 00 00 00h
Access:	Read/Write
Size:	32 bits

This register contains the *next* "ring context ID" associated with the ring buffer.

Programming Notes:

- The current context (RCCID) register can be updated indirectly from this register on a context switch event. Note that this can only be triggered by executing an MI_ARB_CHECK command in the current context or if the current context runs dry (head pointer becomes equal to tail pointer).

Bit	Description

31:12	**Logical Ring Context Address (LRCA):**
	This field contains the 4 KB-aligned Memory address of the next Ring Context Descriptor associated with this ring buffer.
	Format = GlobalGraphicsVirtualAddress[31:12]

11:1	Reserved: MBZ
0	**Valid:** 1 = The other field of this register is valid. A valid ring context is pointed at by the LRCA field of this register. 0 = The other field of this register is invalid. No next context is available to run should the current one execute MI_ARB_CHECK or run out of instructions. This bit is reset by HW when the current context ends and the "next" context becomes the current one. Once that happens, SW may submit a new "next" context.

8.19.3 Registers in the VCE Cryptal Engine [DevCTG+]

8.19.3.1 CRYP_VIDEO_KEY_FRESHNESS – Crypto Key Exchange Read Register

This Read Only command communicates the 128bit freshness value which qualifies the session key used in the content encryption/decryption. Note that the actual modification and incorporation of this new freshness value into the current session key takes effect only on receiving the MFX_CRYPTO_KEY_EXCHANGE command with the 'Key Use Indication' field directing the use of new Freshness value.

The Key Freshness information is requested by the driver, to pass it on to the app. The driver will generate the first DWord read command and will follow it up with three more reads with the address indexed, to get a Dword for each read, starting from LSDword to MSDword.

A new **set** of four read commands will return a new 128bit freshness value.

Address offset:	0441**0**-0441**3**h 0441**4**-0441**7**h 0441**8**-0441**B**h 0441**C**-0441**F**h
After Reset:	00000000h
Normal Access:	Read only

Bit	Description
31:0	**32bits each of the Key Freshness value of 128bits.** The specific DWord selected for read, out of the 4 DWords, is based on the **index value** (the **MS 2bits** of the **highlighted nibble**) in the address. It is expected that the driver will read these DWord in order, starting with DWord_0 and ending with Dword_3.

8.19.3.2 CRYP_PACKET – Crypto Encrypted packet Read Register

This Read Only command is executable **only in test mode**. It allows the test mode setup to read out 16bytes of encrypted internally stored information, using the MMIO Read interface.

This register is only accessible when Text mode bit is not set.

Address offset: 0442**0**-0442**3**h
0442**4**-0442**7**h
0442**8**-0442**B**h
0442**C**-0442**F**h

After Reset: 00000000h
Normal Access: Read only

Bit	Description
31:0	**32bits each of the Encrypted result value of 128bits.** The specific DWord selected for read, out of the 4 DWords, is based on the **index value** (the **MS 2bits** of the **highlighted nibble**) in the address.

8.19.3.3 CRYP_DISPLAY_PIPE_TO_PORT_STATUS – Crypto Display Pipe to Port Status Read Register

This Read Only command communicates to the driver, the 128bit encrypted status of which are enabled and connected to the two internal Display pipes. This information is requested by the App. through the driver, to determine if any spurious enabling of a port to a pipe has happened. The driver will generate the first DWord read command and will follow it up with three more reads with the address indexed, to get a Dword for each read, starting from LSDword to MSDword.

This encrypted read status block will be computed new registers are updated (programmed) and also when a new MFX_CRYPTO_KEY_EXCHANGE command is received. The key used for encryption will be the current session key (when these status bits get programmed) and the updated and new key (after a new MFX_CRYPTO_KEY_EXCHANGE command is executed). The encryption mode used is AES-128 ECB.

The 16byte status to be encrypted[127:0] := “DISPLAY_PIPE_TO_PORT_STATUS[15:0]” & “All 0s”[111:0].

Address offset:	0443**0**-0443**3**h 0443**4**-0443**7**h 0443**8**-0443**B**h 0443**C**-0443**F**h
After Reset:	00000000h
Normal Access:	Read only

Bit	Description
31:0	**32bits each of the Encrypted DISPLAY_PIPE_TO_PORT_STATUS value of 128bits.** The specific DWord selected for read, out of the 4 DWords, is based on the **index value** (the **MS 2bits** of the **highlighted nibble**) in the address. Note: Before the encryption, the status bit definition in the most significant Dword (Dword_3): The 12bits of pipe status mapping: Display_Pipe**A**_P2P[7:0] & Display_Pipe**B**_P2P[7:0], where the individual port mapping for each pipe is: Bit_7: ‘1’ D enabled, ‘0’ – Disabled Bit_6: ‘1’ –C enabled, ‘0’ – Disabled, Bit_5: ‘1’ –B enabled, ‘0’ – Disabled, Bit_4: ‘1’ - TV Out port enabled, ‘0’ – Disabled, [CLN] only Bit_3: ‘1’ C port enabled, ‘0’ – Disabled, Bit_2: ‘1’ - B port enabled, ‘0’ – Disabled Bit_1: ‘1’ – LVDS port enabled, ‘0’ – Disabled Bit_0: ‘1’ - Analog (CRT) A enabled, ‘0’ – Disabled After the decryption on the driver end, the status bits are in the MSB positions. The rest of all the four Dword bits after decryption should be ‘0’s, and the hw should decode for this.

8.20 Interrupt Control Registers

The Interrupt Control Registers described below all share the same bit definition. The bit definition is as follows:

Table 8-3. Bit Definition for Interrupt Control Registers

Bit	Description
31:4	**Reserved. MBZ:** These bits may be assigned to interrupts on future products/steppings.
7	**Timeout Counter Expired:** Set when the VCS timeout counter has reached the timeout thresh-hold value.
6:4	**Reserved:** MBZ
3	**Reserved:** MBZ
2	**Render Command Parser Master Error:** When this status bit is set, it indicates that the hardware has detected an error. It is set by the device upon an error condition and cleared by a CPU write of a one to the appropriate bit contained in the Error ID register followed by a write of a one to this bit in the IIR. Further information on the source of the error comes from the "Error Status Register" which along with the "Error Mask Register" determines which error conditions will cause the error status bit to be set and the interrupt to occur. **Page Table Error:** Indicates a page table error. **Instruction Parser Error**: The Renderer Instruction Parser encounters an error while parsing an instruction.
1	**Sync Status:** This bit is toggled when the Instruction Parser completes a flush with the sync enable bit active in the INSTPM register. The toggle event will happen after all the graphics engines are flushed. The HW Status DWord write resulting from this toggle will cause the CPU's view of graphics memory to be coherent as well (flush and invalidate the render cache).
0	**Render Command Parser User Interrupt:** This status bit is set when an MI_USER_INTERRUPT instruction is executed on the Render Command Parser. Note that instruction execution is not halted and proceeds normally. A mechanism such as an MI_STORE_DATA instruction is required to associate a particular meaning to a user interrupt.

8.20.1.1 BCS_IPEIR—Instruction Parser Error Identification Register (Debug)

Address Offset:	04064h–04067h
Default Value:	0000 0000h
Access:	Read Only
Size:	32 bits

The IPEIR register identifies the general location of instructions that generated an Invalid Instruction Errors for the Renderer IP. (Note: The header (DWord 0) of the offending instruction will be stored in the IPEHR register).

Bit	Description
31:4	Reserved. Read as zero
3	**Batch Buffer Error:** If this bit is set the faulting instruction was executed from a batch buffer. If this bit is clear the faulting instruction was executed directly from a ring buffer.
2:0	Reserved. Read as zero

8.20.1.2 BCS_IPEHR—Instruction Parser Error Header Register (Debug)

Address Offset:	04068h–0406Bh
Default Value:	0000 0000h
Access:	Read Only
Size:	32 bits

The IPEHR register is used to identify the instructions that generate Invalid Instruction Errors. This register is loaded with the header (DWord 0) of each instruction that is executed. It will therefore hold the header of an instruction that generates an Invalid Instruction Error.

Bit	Description
31:0	**Header:** This field will contain the header (DWord 0) of a Media Decode IP instruction that generates an Invalid Instruction Error.

8.20.1.3 BCS_ACTHD — Active Head Pointer Register (Debug)

Address Offset:	04074h–04077h
Default Value:	0000 0000h
Access:	Read Only
Size:	32 bits

This register contains the Head “Pointer” (DWord Graphics Memory Address) of the ring buffer.

Bit	Description
31:2	**Head Pointer:** DWord Graphics Address corresponding to the Head Pointer of the ring or batch buffer.
1:0	Reserved: MBZ

8.20.1.4 BCS_DMA_FADD —DMA Engine Fetch Address (Debug)

Address Offset:	04078h – 0407Bh
Default Value:	0000 0000h
Access:	Read Only
Size:	32 bits

This register contains the QWord offset from the start address of the instruction being fetched by the DMA engine.

Bit	Description
31:3	**Current DMA QWord Offset:** This field contains the offset of the QWord (from the start of the ring buffer or batch buffer) that the “Video Decode” instruction parser DMA engine is currently accessing (fetching). Note that this offset will typically lead the Head offset (as instructions must be fetched before execution).
2:0	Reserved: MBZ

8.20.1.5 BCS_HWS_PGA — Hardware Status Page Address Register

Address Offset:	04080h–04083h
Default Value:	1FFF F000h
Access:	Read/Write
Size:	32 bits

This register is used to program the 4 KB-aligned System Memory address of the Hardware Status Page used to report hardware status into (typically cacheable) System Memory.

Bit	Description
31:12	**Address:** This field is used by SW to specify Bits 31:12 of the 4 KB-aligned System Memory address of the 4 KB page known as the “Hardware Status Page”. Bits 11:0 of the address MBZ. Format = Bits 31:12 of Graphics Memory Address
11:0	Reserved: MBZ

The following table defines the layout of the Hardware Status Page:

DWord Offset	Description
3:0	**Reserved.** Must not be used.
4	**Head Pointer Storage:** The contents of the Ring Buffer Head Pointer register (register DWord 1) are written to this location either as result of an MI_REPORT_HEAD instruction or as the result of an “automatic report” (see RINGBUF registers).
0Fh:05h	**Reserved.** Must not be used.
(3FFh –	These locations can be used for general purpose via the MI_STORE_DATA_INDEX or

DWord Offset	Description
010h)	MI_STORE_DATA_IMM instructions.

8.20.1.6 BCS_NOPID — NOP Identification Register

Address Offset:	04094h–04097h
Default Value:	0000 0000h
Access:	Read Only
Size:	32 bits

The BCS_NOPID register contains the Noop Identification value specified by the last MI_NOOP instruction that enabled this register to be updated.

Bit	Description
31:22	Reserved: MBZ
21:0	**Identification Number:** This field contains the 22-bit Noop Identification value specified by the last MI_NOOP instruction that enabled this field to be updated.

8.20.1.7 BCS_MI_MODE — Mode Register for Software Interface

Address Offset:	0409Ch–0409Fh
Default Value:	0000 0000h
Access:	Read/Write
Size:	32 bits

The MI_MODE register contains information that controls software interface aspects of the command parser.

Bit	Description
31:16	**Masks:** A “1” in a bit in this field allows the modification of the corresponding bit in Bits 15:0
15:12	**Reserved** Read/Write
11	**Invalidate UHPTR enable:** If bit set H/W clears the valid bit of BCS_UHPTR (4134h, bit 0) when current active head pointer is equal to UHPTR.
10	**Reserved** Read/Write
9	**Ring Idle (Read Only Status bit)** 0 = Parser not Idle 1 = Parser Idle ***Writes to this bit are not allowed.***
8	**Stop Ring** 0 = Normal Operation. 1 = Parser is turned off.

Bit	Description
	Software must set this bit to force the Ring and Command Parser to Idle. Software must read a "1" in Ring Idle bit after setting this bit to ensure that the hardware is idle. ***Software must clear this bit for Ring to resume normal operation.***
7:2	**Reserved** Read/Write
1	**Dummy Read Disable.** Nominally a command stream flush is completed with a dummy read to memory to push all pending writes. Setting this bit to a "1" disables the dummy read.
0	**Reserved** Read/Write

8.20.1.8 BCS_INSTPM—Instruction Parser Mode Register

Address Offset:	040C0h–040C3h
Default Value:	0000 0000h
Access:	Read/Write
Size:	32 bits

The BCS_INSTPM register is used to control the operation of the BCS Instruction Parser. Certain classes of instructions can be disabled (ignored) – often useful for detecting performance bottlenecks. Also, "Synchronizing Flush" operations can be initiated – useful for ensuring the completion (vs. only parsing) of rendering instructions.

Programming Notes:

All Reserved bits are implemented.

Bit	Description
31:16	**Masks:** These bits serve as write enables for bits 15:0. If this register is written with any of these bits clear the corresponding bit in the field 15:0 will not be modified. Reading these bits always returns 0s.
15:6	Reserved: MBZ
5	**Sync Flush Enable:** This field is used to request a Sync Flush operation. The device will automatically clear this bit before completing the operation. See Sync Flush (*Programming Environment*). **Programming Note:** • The command parser must be stopped prior to issuing this command by setting the **Stop Ring** bit in register **BCS_MI_MODE**. Only after observing **Ring Idle** set in **BCS_MI_MODE** can a Sync Flush be issued by setting this bit. Once this bit becomes clear again, indicating flush complete, the command parser is re-enabled by clearing **Stop Ring**. Format = Enable (cleared by HW)
4:0	Reserved: MBZ

8.20.1.9 BCS_UHPTR — Pending Head Pointer Register

Address Offset:	04134h–04137h
Default Value:	0000 0000h
Access:	Read/Write
Size:	32 bits

Bit	Description
31:3	**Head Pointer Address**: This register represents the GFX address offset where execution should continue in the ring buffer following execution of an MI_ARB_CHECK command. Format = MI_Graphics_Offset
2:1	Reserved: MBZ
0	**Head Pointer Valid:** 1 = Indicates that there is an updated head pointer programmed in this register 0 = No valid updated head pointer register, resume execution at the current location in the ring buffer This bit is set by the software to request a pre-emption. It is reset by hardware after the head pointer in this register is read. The hardware uses the head pointer programmed in this register at the time the reset is generated.

8.20.1.10 BCS_CNTR—Counter for the Bit Stream Decode Engine

Address Offset:	04178h–0417Bh
Default Value:	FFFF FFFFh
Access:	Read/Write
Size:	32 bits

Bit	Description
31:0	**Count Value**: Writing a Zero value to this register starts the counting. Writing a Value of FFFF FFFF to this counter stops the counter

8.20.1.11 BCS_THRSH—Threshold for the Counter of Bit Stream Decode Engine

Address Offset:	0417Ch–0417Fh
Default Value:	00014500h
Access:	Read/Write
Size:	32 bits

Bit	Description
31:0	**Threshold Value**: The value in this register reflects the number of clocks the bit stream decode engine is expected to run. If the value is exceeded the counter is reset and an interrupt may be enabled in the device.

8.20.1.12 BCS_BB_ADDR—Batch Buffer Head Pointer Register

Address Offset: 04140h–04147h
Default Value: 0000 0000 0000 0000h
Access: Read-Only
Size: 64 bits

This register contains the current QWord Graphics Memory Address of the last-initiated batch buffer.

Bit	Description
63:32	Reserved: MBZ
31:3	**Batch Buffer Head Pointer:** This field specifies the QWord-aligned Graphics Memory Address where the last initiated Batch Buffer is currently fetching commands. If no batch buffer is currently active, the Valid bit will be 0 and this field will be meaningless. .
2:1	Reserved: MBZ
0	**Valid:** 1 = Batch buffer Valid 0 = Batch buffer Invalid

8.20.1.13 BCS_RCCID—Ring Buffer Current Context ID Register

Address Offset: 04190h–04193h
Default Value: 00 00 00 00h
Access: Read/Write
Size: 32 bits

This register contains the current “ring context ID” associated with the ring buffer.

Programming Notes:

- The current context registers must not be written directly (via MMIO). The RCCID register should only be updated indirectly from RNCID.

Bit	Description
31:12	**Logical Ring Context Address (LRCA):** This field contains the 4 KB-aligned Memory address of the current Ring Context Descriptor associated with this ring buffer. See the RNCID register for the Descriptor format. Format = GlobalGraphicsVirtualAddress[31:12]
11:1	Reserved: MBZ
0	**Valid:** 1 = The other field of this register is valid. A ring context is executing and the LRCA field contains the address of its context descriptor. 0 = The other field of this register is invalid. No ring context is executing. This streamer is idle or it is being used in Basic Scheduler mode where the ring buffer registers are manipulated directly and no ring context is used.

8.20.1.14 BCS_RNCID—Ring Buffer Next Context ID Register

Address Offset:	04194h–04197h
Default Value:	00 00 00 00h
Access:	Read/Write
Size:	32 bits

This register contains the *next* "ring context ID" associated with the ring buffer.

Programming Notes:

The current context (RCCID) register can be updated indirectly from this register on a context switch event. Note that this can only be triggered by executing an MI_ARB_CHECK command in the current context or if the current context runs dry (head pointer becomes equal to tail pointer).

Bit	Description

31:12	**Logical Ring Context Address (LRCA):**
	This field contains the 4 KB-aligned Memory address of the next Ring Context Descriptor associated with this ring buffer.
	Format = GlobalGraphicsVirtualAddress[31:12]

11:1	Reserved: MBZ
0	**Valid:** 1 = The other field of this register is valid. A valid ring context is pointed at by the LRCA field of this register. 0 = The other field of this register is invalid. No next context is available to run should the current one execute MI_ARB_CHECK or run out of instructions. This bit is reset by HW when the current context ends and the "next" context becomes the current one. Once that happens, SW may submit a new "next" context.

8.21 Frame Buffer Compression Control ([DevCL] Only)

This section describes the registers associated with the Frame Buffer Compression function. The primary motivation of FBC is power savings and thus it is only applicable to the Mobile Product.

Programming Notes:

Frame buffer compression has to be disabled (via FBC_CONTROL[31] = 0), and software has to wait until compression not in progress (FBC_STATUS[31] == 0) before changing any of the following fields:
FBC_CFB_BASE
FBC_LL_BASE
FBC_CONTROL[Mode Select]
FBC_CONTROL[Compressed Frame Buffer Stride]
FBC_CONTROL[Fence Number]

8.21.1 FBC_CFB_BASE — Compressed Frame Buffer Base Address

FBC_CFB_BASE — Compressed Frame Buffer Base Address

Register Type:	MMIO
Address Offset:	3200h
Project:	DevCL
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

This register specifies the physical memory address at which the Compressed Frame Buffer is located. Note that the Compressed Frame Buffers must be in Non Cacheable memory and not relocated while FBC is active.

Bit	Description
31:12	**Compressed Frame Buffer Address** Project: DevCL Default Value: 0h Address: PhysicalAddress[31:12] This register specifies Bits 31:12 of the physical address of the Compressed Frame Buffer. **Programming Notes** Software must guarantee that the Compressed Frame Buffer is stored in contiguous physical memory. The buffer must be 4K byte aligned. This field should not be changed unless FBC is inactive (the first VBlank start after **Enable Frame Buffer Compression** has been cleared.)
11:0	**Reserved** Project: DevCL Format: MBZ

8.21.2 FBC_LL_BASE — Compressed Frame Line Length Buffer Address

FBC_LL_BASE — Compressed Frame Line Length Buffer Address	
Register Type:	MMIO
Address Offset:	3204h
Project:	DevCL
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

This register specifies the physical memory address at which the Compressed Frame Line Length Buffer is located. **Note that the Compressed Frame Buffers must be in Non Cacheable memory and not relocated while FBC is active.**

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:12</td><td>
<table>
<tr><th colspan="3">Compressed Frame Line Length Buffer Address</th></tr>
<tr><td>Project:</td><td colspan="2">DevCL</td></tr>
<tr><td>Default Value:</td><td>0h</td><td></td></tr>
<tr><td>Address:</td><td colspan="2">PhysicalAddress[31:12]</td></tr>
<tr><td colspan="3">This register specifies Bits 31:12 of the physical address of the Compressed Frame Line Length Buffer.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Software must guarantee that the Compressed Frame Line Length Buffer is stored in contiguous physical memory. The buffer must be 4K byte aligned. This field should not be changed unless FBC is inactive (the first VBlank start after <b>Enable Frame Buffer Compression</b> has been cleared.)</td></tr>
</table>
</td></tr>
<tr><td>11:0</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>DevCL</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
</table>

8.21.3 FBC_CONTROL — Frame Buffer Compression Control Register

FBC_CONTROL — Frame Buffer Compression Control Register

Register Type:	MMIO
Address Offset:	3208h
Project:	DevCL
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

This register is used to control the operation of RLE-FBC.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31</td><td>
<table>
<tr><td colspan="3"><b>Enable Frame Buffer Compression</b></td></tr>
<tr><td>Project:</td><td colspan="2">DevCL</td></tr>
<tr><td>Default Value:</td><td>0h</td><td></td></tr>
<tr><td>Format:</td><td>Enable</td><td></td></tr>
<tr><td colspan="3">This bit is used to globally enable or disable the RLE-FBC function (compression and decompression) at the next VBlank start.</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>Disable</td><td>Disable frame buffer compression.</td><td>DevCL</td></tr>
<tr><td>1h</td><td>Enable</td><td>Enable frame buffer compression.</td><td>DevCL</td></tr>
</table>
</td></tr>
<tr><td>30</td><td>
<table>
<tr><td colspan="3"><b>Mode Select</b></td></tr>
<tr><td>Project:</td><td colspan="2">DevCL</td></tr>
<tr><td>Default Value:</td><td>0h</td><td></td></tr>
<tr><td>Format:</td><td>U1</td><td></td></tr>
<tr><td colspan="3"></td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>Single Pass</td><td>Single Pass mode</td><td>DevCL</td></tr>
<tr><td>1h</td><td>Periodic Pass</td><td>Periodic mode</td><td>DevCL</td></tr>
</table>
</td></tr>
</table>

FBC_CONTROL — Frame Buffer Compression Control Register

29:16	**Interval**				
	Project:	DevCL			
	Default Value:	0h			
	Format:	U14			
	Range	[1,16383]			
	This is interval for which the compressor waits between passes. In Periodic Mode this field determines the interval length, in terms of frames (VBlanks). Zero is an illegal value.				
15	**Stop Compressing on Modification (DEBUG ONLY)**	Project:	DevCL	Format:	Enable
	If set to ‘1’ the compressor will abort a subsequent compressing pass when any modification to the source frame buffer is detected.				
14	**Uncompressible Enable**	Project:	DevCL	Format:	Enable
	If set to a ‘1’ the compressor marks as "Uncompressible 10" (see the FBC_TAG register) if any scanline in a pair cannot be compressed. In Default mode Uncompressible mode is turned off.				
13	**Reserved**	Project:	DevCL	Format:	MBZ
12:5	**Compressed Frame Buffer Stride**	Project:	DevCL	Format:	(Stride in 64Byte units) – 1
	This is the stride for the compressed frame buffer. This value is used to determine the line-to-line increment for the compressed frame buffer. Lines that cannot be compressed to a stride size or less are not compressed at all. This field must be set to a value less than or equal to the stride of the source (uncompressed) frame buffer. 00h = 64B stride				
4	**Reserved**	Project:	DevCL	Format:	MBZ
3:0	**Fence Number**	Project:	DevCL	Format:	U3
	This field specifies the FENCE number corresponding to the placement of the uncompressed frame buffer. (Note that only tiled frame buffers can be compressed). This field is double buffered in hardware. Only the host accesses the uncompressed frame buffer using a fence.				

8.21.4 FBC_COMMAND — Frame Buffer Compression Command Register

FBC_COMMAND — Frame Buffer Compression Command Register	
Register Type:	MMIO
Address Offset:	320Ch
Project:	DevCL
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

This register is used to request a frame buffer compression pass while in Single Pass mode.

Bit	Description
31:1	**Reserved** | Project: DevCL | Format: MBZ
0	**Compress Enable** | Project: DevCL | Format: Enable Software can set this bit to trigger compression in Single Pass mode. The compressor clears this bit after the compression pass is completed. This bit is ignored in Periodic Mode (i.e., it will not cause a compression pass and will always read as '0').

8.21.5 FBC_STATUS — Frame Buffer Compression Status Register

FBC_STATUS — Frame Buffer Compression Status Register	
Register Type:	MMIO
Address Offset:	3210h
Project:	DevCL
Default Value:	2000 0000h
Access:	RO / R/W
Size (in bits):	32

This register contains status information associated with the RLE-FBC function. The information is read-only in normal operation, though some fields can be programmed as a TEST MODE.

Bit	Description
31	**Compressing** Project: DevCL Security: RO Default Value: 0h Format: Flag This status bit indicates that the device is currently within a compression pass.

FBC_STATUS — Frame Buffer Compression Status Register

30	**Compressed**	
	Project:	DevCL
	Security:	RO normally, R/W TEST MODE
	Default Value:	0h
	Format:	Flag

This bit indicates that a compressed frame buffer is available at the address contained in the FB_CFB_BASE register.

In normal operation the compressor sets this bit when it has completed the compression pass. During compression this bit is not set.

As a test mode this bit can be set if there is a software-created compressed buffer available at the address in the FB_CFB_BASE register. Test-Mode software must check that compression is **not** in progress before setting this bit. If RLE-FBC is enabled, the compressor will clear this bit when it starts the next recompression pass.

29	**Any Modified**	
	Project:	DevCL
	Security:	RO normally, R/W TEST MODE
	Default Value:	1h
	Format:	Flag

1 = (default) Indicates that the frame buffer has been modified since the last compression pass. The compressor sets this bit on the first write to the frame buffer from the application/driver or upon an allocation within the render cache (e.g., as a result of Blt, 3D or MPEG activity). The fence number and frame buffer base address are used to determine if a write modified the frame buffer. The bit is cleared by the compressor at the start of the next compression pass.

In normal operation this bit is read only (software must not write this bit) and defaults to a **"1"**.

As a test mode this bit can be set if there is a software-created compressed buffer with modified lines available at the address contained the FB_CFB_BASE register. SW must check that compression is **not** in progress before setting this bit. If enabled, the compressor will clear this bit when it initiates the next compression pass. This test mode is used for continuous-mode compression testing.

28:11	**Reserved**	Project:	DevCL	Format:	MBZ

10:0	**Current Line Compressing**	
	Project:	DevCL
	Security:	RO
	Default Value:	0h
	Format:	U11

This read only field indicates the line number that the compressor is currently processing.

If this field is 0 and the **Compressing** bit (Bit 31) is set, the compressor is currently on display frame line 1.

8.21.6 FBC_CONTROL2— Frame Buffer Compression 2nd Control Register

<table>
<tr><th colspan="2">FBC_CONTROL2— Frame Buffer Compression 2nd Control Register</th></tr>
<tr><td><b>Register Type:</b></td><td>MMIO</td></tr>
<tr><td><b>Address Offset:</b></td><td>3214h</td></tr>
<tr><td><b>Project:</b></td><td>DevCL</td></tr>
<tr><td><b>Default Value:</b></td><td>0000 0000h</td></tr>
<tr><td><b>Access:</b></td><td>R/W</td></tr>
<tr><td><b>Size (in bits):</b></td><td>32</td></tr>
<tr><td colspan="2">This register is used to control the operation of RLE-FBC.</td></tr>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:3</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>DevCL</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>4</td><td>
<table>
<tr><th colspan="2">Double Buffer FBC Fence and Fence_DisplayY Offset Register Fields</th></tr>
<tr><td>Project:</td><td>DevCL</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>Disable</td></tr>
<tr><td colspan="2"></td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td></td><td>Double buffer</td><td>DevCL</td></tr>
<tr><td>1h</td><td></td><td>Don’t double buffer</td><td>DevCL</td></tr>
</table>
</td></tr>
<tr><td>3:2</td><td>
<table>
<tr><th colspan="2">FBC C3 Mode</th></tr>
<tr><td>Project:</td><td>DevCL</td></tr>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>U2</td></tr>
<tr><td colspan="2"></td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>00</td><td></td><td>FBC IDLENESS is not looked at in order to enter Self Refresh</td><td>DevCL</td></tr>
<tr><td>01</td><td></td><td>FBC IDLENESS is looked at in order to enter Self Refresh</td><td>DevCL</td></tr>
<tr><td>10</td><td></td><td>FBC IDLENESS is looked at in order to enter Self Refresh. But FBC enters IDLE as it finishes compressing the current scanline pair and enters IDLE as soon as csunit asserts the inc3 signal.</td><td>DevCL</td></tr>
<tr><td>11</td><td>Reserved</td><td>Reserved</td><td>DevCL</td></tr>
</table>
</td></tr>
</table>

FBC_CONTROL2— Frame Buffer Compression 2nd Control Register

1

CPU Fence enable	
Project:	DevCL
Default Value:	0h
Format:	Enable

Value	**Name**	**Description**	**Project**
0h		Display Buffer is not in a CPU fence. No modifications are expected from CPU to the Display Buffer.	DevCL
1h		Display Buffer exists in a CPU fence.	DevCL

0

Frame Buffer Compression Display Plane Select A/B	
Project:	DevCL
Default Value:	0h
Format:	Flag

Value	**Name**	**Description**	**Project**
0h		Enable frame buffer compression on Plane A.	All
1h		Enable frame buffer compression on Plane B.	All

Programming Notes	**Project**
Before changing this bit s/w needs to make sure that FBC is disabled and the “COMPRESSING” bit in the FBC_CONTROL register comes to a “0”.	DevCL

8.21.7 FBC_DISPYOFF — FBC Fence Display Buffer Y Offset

FBC_DISPYOFF — FBC Fence Display Buffer Y Offset

Register Type:	MMIO
Address Offset:	321Bh
Project:	DevCL
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

Bit	Description
31:12	**Reserved** Project: DevCL Format: MBZ
11:0	**Fence_YDisp** Project: DevCL Format: U12 Y offset from the fence to the Display Buffer base

8.21.8 FBC_MOD_NUM— FBC Number of Modifications for Recompression

FBC_MOD_NUM— FBC Number of Modifications for Recompression

Register Type:	MMIO
Address Offset:	3220h
Project:	DevCL
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32
Trusted Type:	1

The purpose of this register is to avoid SR exit unless the programmed number of modifications have been made to the Display buffer.

Bit	Description
31:1	**FBC_Mod_Num** Project: DevCL Format: U12 Number of modifications to the display buffer required before recompression is attempted. If the number of modifications to the Frame Buffer is not equal to the programmed count value at the end of the interval, re-compression is not attempted.
0	**FBC_Mod_Num_Valid** Project: DevCL Format: Flag Only if this bit is set will the above count value be looked at.

8.21.9 FBC_TAG — Frame Buffer Compression TAG Interface (DEBUG)

FBC_TAG — Frame Buffer Compression TAG Interface (DEBUG)

Register Type:	MMIO
Address Offset:	3300h
Project:	All
Default Value:	00000000h;
Access:	R/W
Size (in bits):	49x32
Trusted Type:	1

The device implements 49 DWords of Tag data for RLE-FBC compression. Each DWord contains storage for a 2-bit Tag value associated with a frame buffer line pair.

49 DWords are required to support the required 1536 display lines (= 48 x 32), as an extra DWord may be required due to the alignment of the source (uncompressed) frame buffer. I.e., if the source frame buffer starts on an odd tile line, line 0 corresponds to bit 1 of 3300 (bit 0 is unused) and the 49th DWord may be required. If the source frame buffer starts on an even tile line, line 0 corresponds to bit 0 of 3300.

DWord	Bit	Description
0..48	31:30	**Tag for lines 30&31** Project: All Format: FBC Tag For lines: (DWord) + 30 and (DWord) + 31
	29:28	**Tag for lines 29&28** Project: All Format: FBC Tag For lines: (DWord) + 30 and (DWord) + 31
	27:26	**Tag for lines 27&26** Project: All Format: FBC Tag For lines: (DWord) + 30 and (DWord) + 31
	25:24	**Tag for lines 25&24** Project: All Format: FBC Tag For lines: (DWord) + 30 and (DWord) + 31
	23:22	**Tag for lines 23&22** Project: All Format: FBC Tag For lines: (DWord) + 30 and (DWord) + 31
	21:20	**Tag for lines 21&20** Project: All Format: FBC Tag For lines: (DWord) + 30 and (DWord) + 31

FBC_TAG — Frame Buffer Compression TAG Interface (DEBUG)

	19:18	**Tag for lines 19&18**	Project:	All	Format:	FBC Tag
		For lines: (DWord) + 30 and (DWord) + 31				
	17:16	**Tag for lines 17&16**	Project:	All	Format:	FBC Tag
		For lines: (DWord) + 30 and (DWord) + 31				
	15:14	**Tag for lines 15&14**	Project:	All	Format:	FBC Tag
		For lines: (DWord) + 30 and (DWord) + 31				
	13:12	**Tag for lines 13&12**	Project:	All	Format:	FBC Tag
		For lines: (DWord) + 30 and (DWord) + 31				
	11:10	**Tag for lines 11&10**	Project:	All	Format:	FBC Tag
		For lines: (DWord) + 30 and (DWord) + 31				
	9:8	**Tag for lines 9&8**	Project:	All	Format:	FBC Tag
		For lines: (DWord) + 30 and (DWord) + 31				
	7:6	**Tag for lines 7&6**	Project:	All	Format:	FBC Tag
		For lines: (DWord) + 30 and (DWord) + 31				
	5:4	**Tag for lines 5&4**	Project:	All	Format:	FBC Tag
		For lines: (DWord) + 30 and (DWord) + 31				
	3:2	**Tag for lines 3&2**	Project:	All	Format:	FBC Tag
		For lines: (DWord) + 30 and (DWord) + 31				
	1:0	**Tag for lines 1&0**	Project:	All	Format:	FBC Tag
		For lines: (DWord) + 30 and (DWord) + 31				

FBC_TAG — Frame Buffer Compression TAG Interface (DEBUG)

31:0

Tag for lines DW# + 1&0

Project:	All	
Format:	FBC Tag	See below

For lines: (DWord) + 30 and (DWord) + 31

Value	Name	Description	Project
00	1Modified	At least one of the associated lines was modified since the last compression pass started.	All
01	Uncompresse d	The associated lines are uncompressed and are candidate for compression in the next pass	All
10	Uncompressib le	The associated lines are uncompressible and are not candidate for compression in the next pass.	All
11	Compressed	The associated lines are compressed	All

8.22 Fence Registers

8.22.1 FENCE — Graphics Memory Fence Table Registers

FENCE — Graphics Memory Fence Table Registers

Register Type:	MMIO
Address Offset:	3000h
Project:	All
Default Value:	00000000h;
Access:	R/W
Size (in bits):	16x64
Trusted Type:	1

Address Offset: 03000h – 03007h: FENCE_0

:

:

0307Ch – 0307Fh: FENCE_15

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

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

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

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

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

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

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

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

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

FENCE — Graphics Memory Fence Table Registers

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0..15</td><td>63:44</td><td>
<table>
<tr><th colspan="2">Fence Upper Bound</th></tr>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Address:</td><td>GraphicsAddress[31:12]</td></tr>
<tr><td colspan="2">Bits 31:12 of the ending Graphics Address of the fence region. Fence regions must be aligned to a 4KB page. This address represents the last 4KB page of the fence region (Upper Bound is included in the fence region).
Graphics Address is the offset within GMADR space.</td></tr>
</table>
</td></tr>
<tr><td></td><td>45:32</td><td>
<table>
<tr><td><b>Reserved</b></td><td>Project:</td><td>All</td><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>31:12</td><td>
<table>
<tr><th colspan="2">Fence Lower Bound</th></tr>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Address:</td><td>GraphicsAddress[31:12]</td></tr>
<tr><td colspan="2">Bits 31:12 of the starting Graphics Address of the fence region. Fence regions must be aligned to 4KB. This address represents the first 4KB page of the fence region (Lowe Bound is included in the fence region).
Graphics Address is the offset within GMADR space.</td></tr>
</table>
</td></tr>
<tr><td></td><td>11:2</td><td>
<table>
<tr><th colspan="3">Fence Pitch</th></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Default Value:</td><td>0h</td><td>DefaultVaueDesc</td></tr>
<tr><td>Format:</td><td>U10-1</td><td>Width in 128 bytes</td></tr>
<tr><td colspan="3">This field specifies the width (pitch) of the fence region in multiple of “tile width”. For Tile X this field must be programmed to a multiple of 512B (“003” is the minimum value) and for Tile Y this field must be programmed to a multiple of 128B (“000” is the minimum value).
000h = 128B
001h = 256B
…
3FFh = 128KB</td></tr>
</table>
</td></tr>
<tr><td></td><td>1</td><td>
<table>
<tr><th colspan="3">Tile Walk</th></tr>
<tr><td>Project:</td><td colspan="2">All</td></tr>
<tr><td>Format:</td><td>MI_TileWalk</td><td></td></tr>
<tr><td colspan="3">This field specifies the spatial ordering of QWords within tiles.</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>MI_TILE_XMAJOR</td><td>Consecutive SWords (32 Bytes) sequenced in the X direction</td><td>All</td></tr>
<tr><td>1h</td><td>MI_TILE_YMAJOR</td><td>Consecutive OWords (16 Bytes) sequenced in the Y direction</td><td>All</td></tr>
</table>
</td></tr>
</table>

FENCE — Graphics Memory Fence Table Registers

0

Fence Valid		
Project:	All	
Format:	MI_ FenceValid	
This field specifies whether or not this fence register defines a fence region.		

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

8.23 GFX MMIO – MCHBAR Aperture

Address Offset:	10000h – 13FFFh
Default Value:	Same as MCHBAR
Access:	Aligned Word, Dword or Qword Read/Write

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

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

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

§§