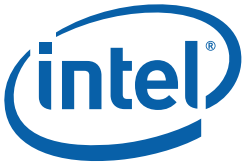


Intel® 965 Express Chipset Family and Intel® G35 Express Chipset Graphics Controller PRM

Programmer's Reference Manual (PRM)

Volume 1: Graphics Core

January 2008

Revision 1.0a

Technical queries: ilg@linux.intel.com

www.intellinuxgraphics.org

Creative Commons License

You are free:

to Share — to copy, distribute,display, and perform the work

Under the following conditions:

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

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

Contents

Figures

Tables

Revision History

Document Number	Revision Number	Description	Revision Date
1	1.0a	Initial release.	January 2008

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.

Note: The term "Gen4" is used throughout the PRM to refer to the Generation 4 family of graphics devices. The devices listed in Table 1-1 are Gen4 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**]

NOTES:

1. Unless otherwise specified, the information in this document applies to all of the devices mentioned in Table 1-1. For Information that does not apply to all devices, the Device Tag is used.
2. Throughout the PRM, references to "All" in a project field refters to all devices in Table 1-1.
3. 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.
4. 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 and G35 Chipset Express Chipset. Familiarity with 2D and 3D graphics programming is assumed.

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

- **PRM, Volume 1: 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, WMV9, 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 treat 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 Gen4 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 Gen4 programs. The Data Port contains the render cache and the constant cache and performs all memory accesses requested by Gen4 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 Gen4 SIMD instruction.
Destination Width	—	The size of each of (possibly) many elements of the destination of a Gen4 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 Gen4 SIMD instruction.
Execution Size	ExecSize	Execution Size indicates the number of data elements processed by a GEN4 SIMD instruction. It is one of the GEN4 instruction fields and can be changed per instruction.
Execution Unit	EU	Execution Unit. An EU is a multi-threaded processor within the GEN4 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 GEN4 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 Gen4 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.
GEN4 Core	—	Alternative name for an EU in the GEN4 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. 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 Gen4 region-based GRF access.
Immediate floating point vector	VF	A numerical data type of 32 bits, an immediate floating point vector of type VF contains 4 floating point elements with 8-bit each. The 8-bit floating point element contains a sign field, a 3-bit exponent field and a 4-bit mantissa field. It may be used to specify the type of an immediate operand in an instruction.

Term	Abbr.	Definition
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. It is required to support WMV9 Profile B video decoder acceleration.
Instruction	—	Data in memory directing an EU operation. Instructions are fetched from memory, stored in a cache and executed on one or more Gen4 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 GEN4 ISA describes the instructions supported by a GEN4 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 both MPEG-2 and WMV.
Jitter	—	Just-in-time compiler.
Kernel	—	A sequence of Gen4 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 Gen4 instructions.
Least Significant Bit	LSB	Least Significant Bit
MathBox	—	See Extended Math Unit
Media	—	Term for operations such as video decode and encode that are normally performed by the Media pipeline.

Term	Abbr.	Definition
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 GEN4 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.
Out-of-loop De-Blocking Filter	OLDB	The de-blocking filter operation outside the decoding loop. It is required to support WMV9 Profile A video decoder acceleration.
Out-of-loop De-Ringing Filter	OLDR	The de-ringing filter operation outside the decoding loop. It is required to support WMV9 Profile A video decoder acceleration.
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.

Term	Abbr.	Definition
Point	—	A drawing object characterized only by position coordinates and width.
Primitive	—	Synonym for object: triangle, rectangle, line or point.
Primitive Topology	—	A composite primitive such as a triangle strip, or line list. Also includes the objects triangle, line and point as degenerate cases.
Provoking Vertex	—	The vertex of a primitive topology from which vertex attributes that are constant across the primitive are taken.
Quad Quad word (QQword)	QQ	A fundamental data type, QQ represents 32 bytes.
Quad Word (QWord)	QW	A fundamental data type, QW represents 8 bytes.
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 Gen4 program that is supplied by the application in an high level shader language, and translated to Gen4 instructions by the jitter.

Term	Abbr.	Definition
Shared Function	SF	Function unit that is shared by EUs. EUs send messages to shared functions; they consume the data and may return a result. The Sampler, Data Port and Extended Math unit are all shared functions.
Shared Function ID	SFID	Unique identifier used by kernels and shaders to target shared functions and to identify their returned messages.
Single Instruction Multiple Data	SIMD	The term SIMD can be used to describe the kind of parallel processing architecture that exploits data parallelism at instruction level. It can also be used to describe the instructions in such architecture.
Source	—	Describes an input or read operand
Spawn	—	To initiate a thread for execution on an EU. Done by the thread spawner as well as most FF units in the 3D pipeline.
Sprite Point	—	Point object using full range texture coordinates. Points that are not sprite points use the texture coordinates of the point's center across the entire point object.
State Descriptor	—	Blocks in memory that describe the state associated with a particular FF, including its associated kernel pointer, kernel resource allowances, and a pointer to its surface state.
State Register	SR	The read-only registers containing the state information of the current thread, including the EUID/TID, Dispatcher Mask, and System IP.
State Variable	SV	An individual state element that can be varied to change the way given primitives are rendered or media objects processed. On Gen4 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.
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 Gen4 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

Term	Abbr.	Definition
Surface State Base Pointer	—	Base address used when referencing binding table and surface state data.
Synchronized Root Thread	—	A root thread that is dispatched by TS upon a 'dispatch root thread' message.
System IP	SIP	There is one global System IP register for all the threads. From a thread's point of view, this is a virtual read only register. Upon an exception, hardware performs some bookkeeping and then jumps to SIP.
System Routine	—	Sequence of Gen4 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 GEN4 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 GEN4 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

Term	Abbr.	Definition
URB Entry	UE	URB Entry: A logical entity stored in the URB (such as a vertex), referenced via a URB Handle.
URB Entry Allocation Size	—	Number of URB entries allocated to a Fixed Function unit.
URB Fence	Fence	Virtual, movable boundaries between the URB regions owned by each FF unit.
URB Handle	—	A unique identifier for a URB entry that is passed down a pipeline.
URB Reference	—	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. GEN4 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.
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 Gen4 region-based GRF access.

Term	Abbr.	Definition
Video Front End	VFE	The first fixed function in the GEN4 generic pipeline; performs fixed-function media operations.
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.

§§

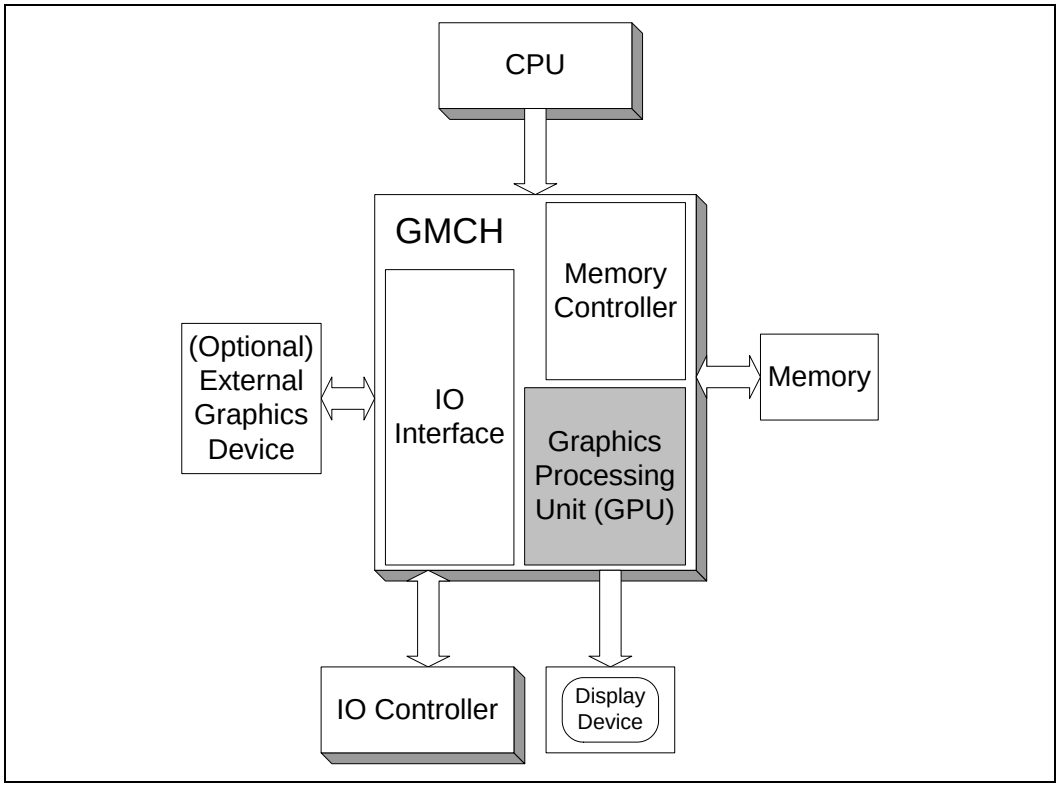
2 Graphics Device Overview

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 Gen4 family GMCH) the 3D/Media engine. This latter piece is made up of the 3D and media "fixed function" (FF) pipelines, and the Gen4 subsystem, which these pipelines make use of to run "shaders" and kernels.

Figure 2-1. GMCH Block Diagram

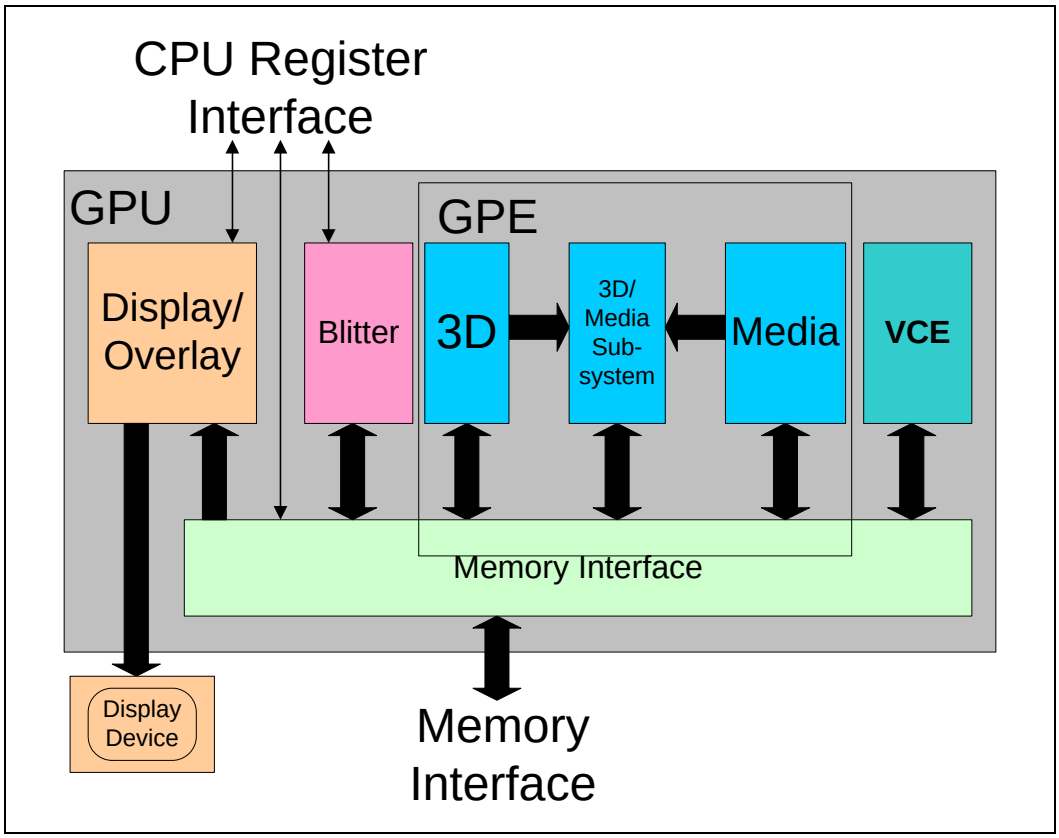


2.2 Graphics Processing Unit (GPU)

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 Gen4 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 GEN4 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 GEN4 GPU. It consists of the 3D and Media fixed-function pipelines, the Command Streamer (CS) unit that feeds them, and the GEN4 Subsystem that provides the bulk of the computations required by the pipelines.

3.2.1 Block Diagram

Figure 3-1. The Graphics Processing Engine

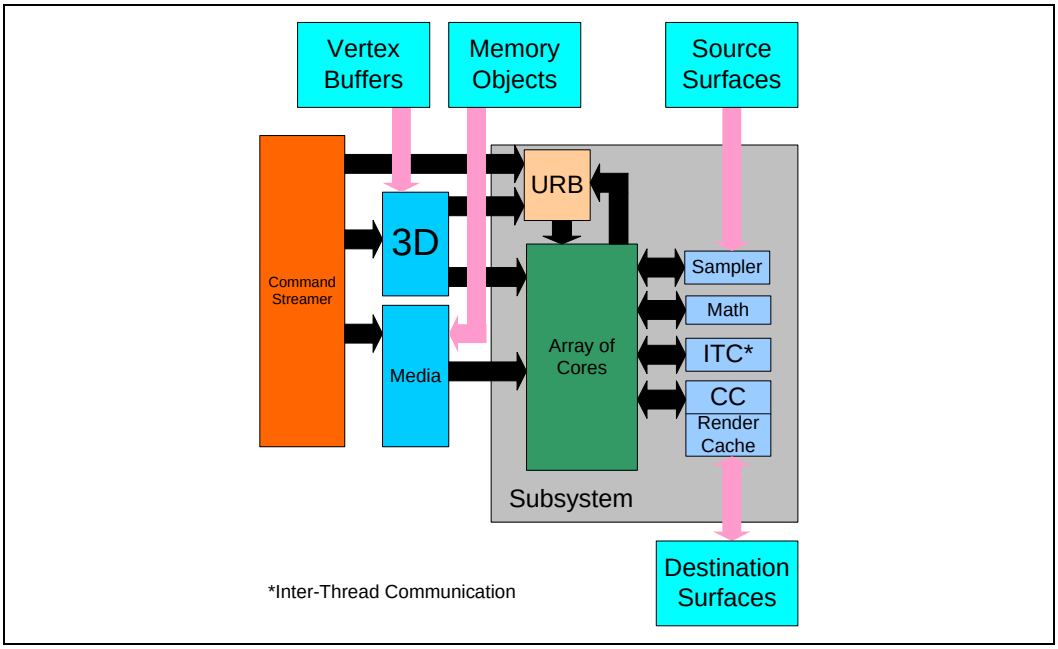
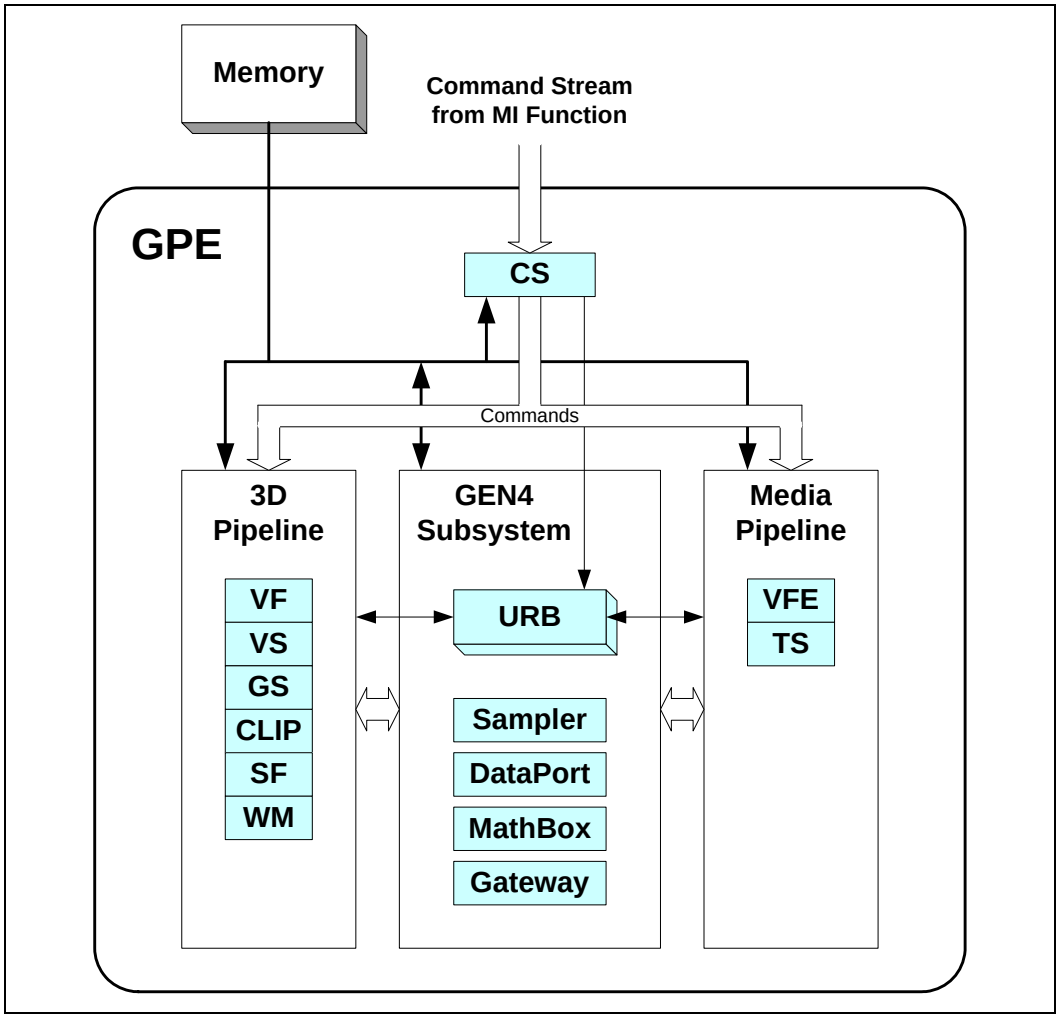


Figure 3-2. GPE Diagram Showing Fixed/Shared Functions



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 GEN4 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 GEN4 threads requested by the VFE unit or as required when the pipeline is used for general processing. See *Media Pipeline Overview*.

3.2.5 GEN4 Subsystem

The GEN4 Subsystem is the collective name for the GEN4 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 GEN4 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
All	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.

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

Table 3-1. Gen4 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	SFID_DP_READ	DataPort Read	Reserved	---
0x5	SFID_DP_WRITE	DataPort Write	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.

Note: ***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

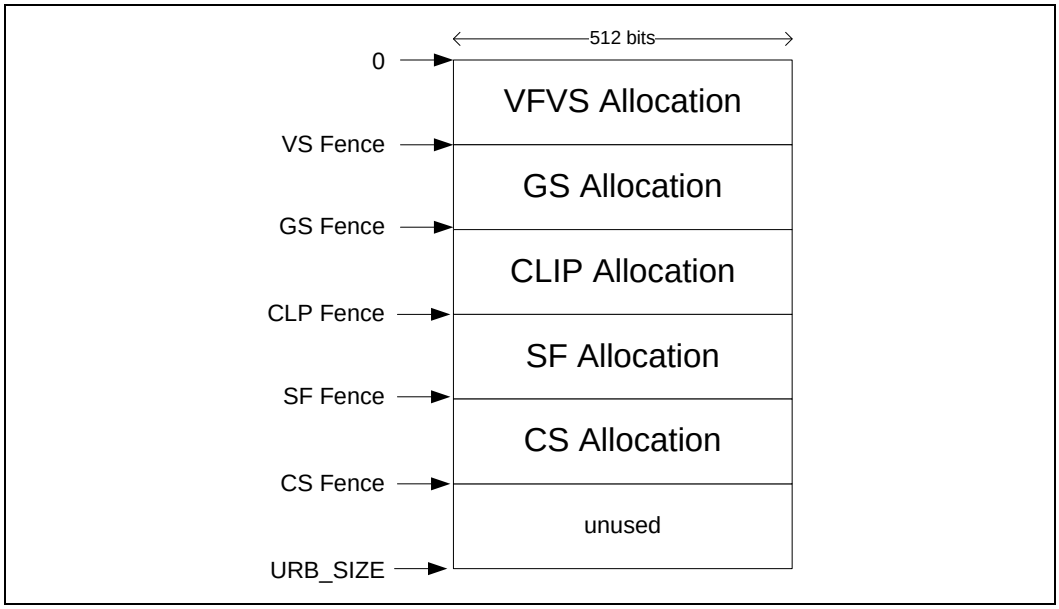
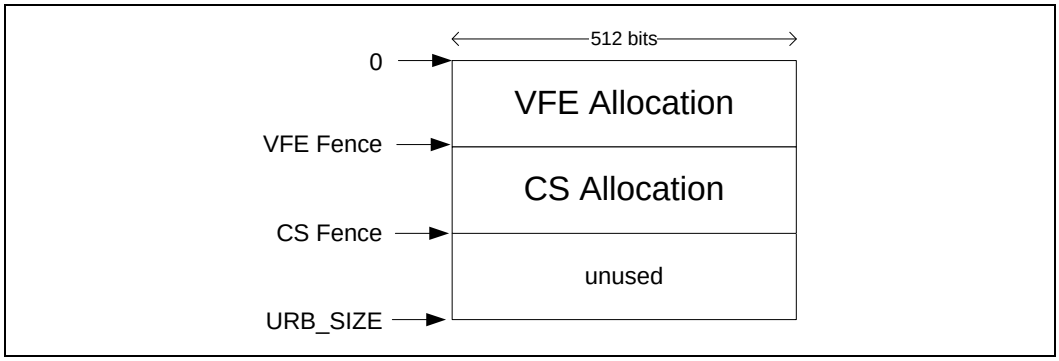


Figure 3-4 URB Allocation – Media Pipeline



Programming Notes:

1. URB Size
    a. URB_SIZE is 16KB = 256 512-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			
Project:	All	**Length Bias:**	2
This command is used to set the fences between URB regions owned by the fixed functions.			
DWord	**Bit**	**Description**	
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode	

URB_FENCE

	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: 00h URB_FENCE Format: OpCode
	15:14	**Reserved** Project: All Format: MBZ
	13	**CS Unit URB Reallocation Request** Project: All Format: Enable FormatDesc If set, the CS unit will perform a URB entry deallocation/reallocation action. **Note:** 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 following description applies to all URB Reallocation Request bits): 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 **Fence** values, or (b) the **Number of URB Entries** or **URB Entry Allocation Size** state variables associated with this unit have been modified since the last reallocation action. Software is required to set this bit accordingly. Within the context's command stream, this is the only cause of a reallocation action --- a reallocation action is **not** 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. Note that all **Fence** 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 **CS Fence** is being changed, the **CLIP, GS, VS and SF Fence** values must be programmed to their correct (previous) values.
	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)

URB_FENCE

	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
	29:20	**CLIP Fence** Project: All Format: U10 representing the first 512-bit URB address beyond this unit's URB space FormatDesc Range [GS Fence,256] 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 [VS Fence,256] 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.

URB_FENCE

	9:0	**VS Fence** Project: All Format: U10 representing the first 512-bit URB address beyond this unit's URB space FormatDesc Range [0,256] 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
	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) 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 [0,256] 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 [CLIP Fence,256] 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

CS_URB_STATE			
Project:	All	**Length Bias:**	2

The CS_URB_STATE packet is used to define the number and size of CURBEs contained within the CS unit's allocated URB region.

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: 01h CS_URB_STATE 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	31:9	**Reserved** Project: All Format: MBZ
	8:4	**URB Entry Allocation Size** Project: All Format: U5 count (of 512-bit units) – 1 FormatDesc Range [0,31] = [1,32] 512-bit units = [2,64] 256-bit URB rows Specifies the length of each URB entry owned by the CS unit.
	3	**Reserved** Project: All Format: MBZ
	2:0	**Number of URB Entries** Project: All Format: U3 count of entries FormatDesc Range [0,4] Specifies the number of URB entries that are used by the CS unit.

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.
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All

CONSTANT_BUFFER		
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. (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.

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.

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

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

STATE_BASE_ADDRESS

3 — 31:12 — **Indirect Object Base Address**

Project: All

Format: GraphicsAddress[31:12] FormatDesc

Specifies the 4K-byte aligned base address for indirect object load in MEDIA_OBJECT command. See Table 3-2 for details on where this base address is used.

11:1 — **Reserved** Project: All Format: MBZ

0 — **Modify Enable**

Project: All

Format: Enable FormatDesc

The address in this dword is updated only when this bit is set.

Value	Name	Description	Project
0h	Disable	Ignore the updated address	All
1h	Enable	Modify the address	All

4 — 31:12 — **General State Access Upper Bound**

Project: All

Format: GraphicsAddress[31:12] FormatDesc

Specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address for general state accesses. This includes all accesses that are offset from **General State Base Address** (see Table 3-2). Read accesses from this address and beyond will return UNDEFINED values. Data port writes to this address and beyond will be "dropped on the floor" (all data channels will be disabled so no writes occur). Setting this field to 0 will cause this range check to be ignored.

If non-zero, this address must be greater than the **General State Base Address**.

11:1 — **Reserved** Project: All Format: MBZ

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

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 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).

3.7.1 STATE_PREFETCH

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

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

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 0h GFXPIPE_COMMON Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h GFXPIPE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 03h STATE_PREFETCH Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:6	**Prefetch Pointer** Project: All Format: GraphicsAddress[31:6] 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.

3.8 System Thread Configuration

3.8.1 STATE_SIP

STATE_SIP			
Project:	All	**Length Bias:**	2

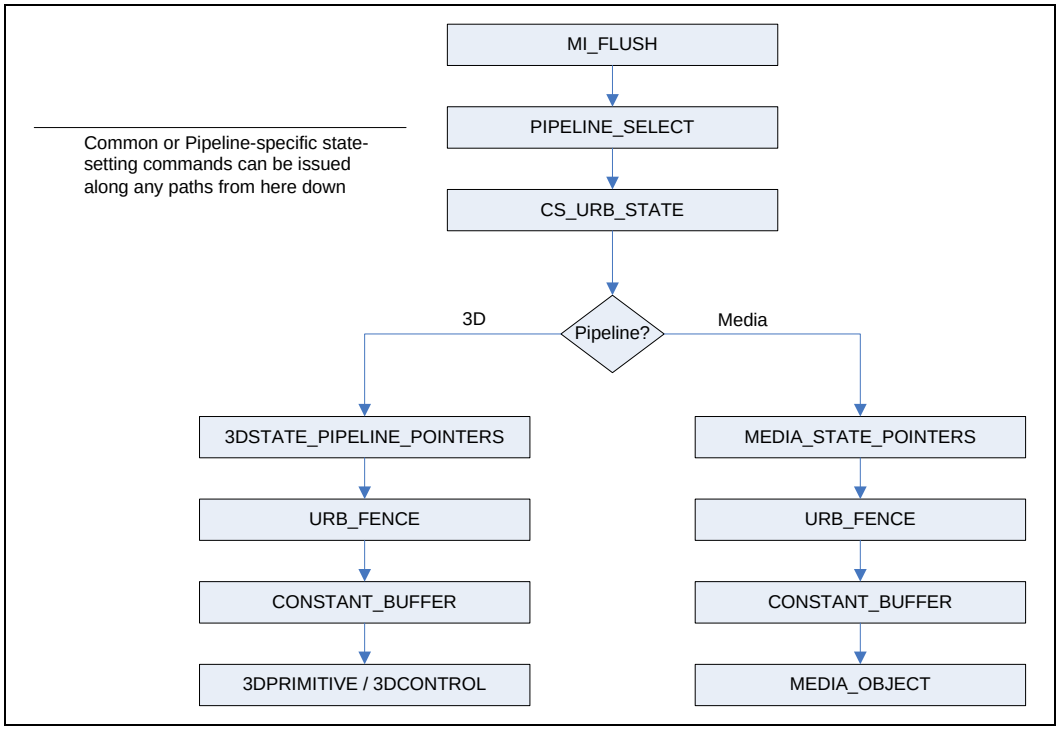
The STATE_SIP command specifies the starting instruction location of the System Routine that is shared by all threads in execution.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 0h GFXPIPE_COMMON Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h GFXPIPE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 02h STATE_SIP Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2 Project: All
1	31:4	**System Instruction Pointer (SIP)** Project: All Format: GeneralStateOffset[31:4] FormatDesc Specifies the instruction address of the system routine associated with the current context as a 128-bit granular offset from the **General State Base Address**. SIP is shared by all threads in execution. The address specifies the double quadword aligned instruction location. **Errata** **Description** **Project** **BWT007** 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.) **[DevBW-A]**
	3:0	**Reserved** Project: All Format: MBZ

3.9 Command Ordering Rules

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

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



3.9.1 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.

3.9.2 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.

3.9.3 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.

3.9.4 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).

3.9.5 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_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.

3.9.6 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.

3.9.7 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.

3.9.8 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.

3.9.9 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.

3.9.10 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 Gen4 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

4.1.5 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 Re-served	15:0 DWord Count	
2D	010	Opcode			Command Dependent Data 4:0 – DWord Count	

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

NOTES:

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

Table 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	110	Opcode: 0h – 1h		DWord Count
Reserved	011	10	110	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
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	
08-09h	Reserved	
0Ah	MI_BATCH_BUFFER_END	
0Bh–0Fh	Reserved	
2+ DWord		
10h	Reserved	
11h	MI_OVERLAY_FLIP	
12h	MI_LOAD_SCAN_LINES_INCL	
13h	MI_LOAD_SCAN_LINES_EXCL	
14h	MI_DISPLAY_BUFFER_INFO	
15h	Reserved	
16h	Reserved	
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	Reserved	
24h	MI_STORE_REGISTER_MEM	

Opcode (28:23)	Command	Comments
25h	Reserved	
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	

Opcode (28:22)	Command	Comments
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	Reserved	n/a
1h	0h	03h-0Ah	Reserved	n/a
1h	0h	0Bh	Reserved	n/a
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

Pipeline Type (28:27)	Opcode	Sub Opcode	Command	Definition Chapter
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–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.

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

Note: 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***
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** **Software Visible 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***

Start Offset	End Offset	Description
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
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

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
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	R/W
020C8h–020CBh	PGTBL_STR2	Page Table Steer Register (Per Process)	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–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

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
02128h–02133h	—	Reserved	—
02134h–02137h	UHPTR	Pending Head Pointer Register	R/W
02138h–0213Fh	—	Reserved	—
02140h–02147h	BB_ADDR	Batch Buffer Current Address	RO
02148h–0214Bh	BB_STATE	Batch Buffer State Register	R/W
0214Ch–0216Fh	—	Reserved	—
02170h–02177h	GFX_FLSH_CNTL	Graphics Flush Control	R/W
02178h–0217Bh	—	Reserved	—
0217Ch–0217Fh	PR_CTR_THRSH	Reserved	—
02180h–02183h	CCID0	Current Context ID 0 (assoc w/ PRB0)	R/W
02184h–0218Fh	—	Reserved	—
02190h–02193h	—	Reserved	
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-0234Fh	—	Reserved	—
02350h-02357h	PS_DEPTH_COUNT	Reported Pixels Passing Depth Test Counter	R/W
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	—

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
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–0251Fh	—	Reserved	—
02520h–02520	GFX_MODE	Graphics Mode	R/W
Probe List Control (02600h–026FFh) : Reserved			
02600h–026FFh	—	Reserved	—
Run List Control (02700h–027FFh) : Reserved			
02700h–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	—
03300h–033C3h	FBC_TAG	Frame Buffer Compression Tag Interface (Debug)	R/W
03400h–03FFFh	—	Reserved	—
Frame Buffer Compression Control (03200h–03FFFh) : Reserved			
03200h–03FFFh	DPFC_CB_BASE	DPFC Compressed Buffer Base Address	R/W

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
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	—
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	—
(04400h–044FFh) : Reserved			
04400h–044FFh	—	Reserved	—

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
MFC Status Registers (04500h–04FFFh) : Reserved			
04500h-04FFFh	—	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	—
VSC Registers (05000h – 05FFFh) : Reserved			
05000h-0506Fh	—	Reserved	—
VSC Registers : Reserved			
05070h-05083h		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

Address Offset	Symbol	Register Name	Access
06018h–0601Bh	DPLLB_CTRL	Display PLL B Control	R/W
0601Ch–0601Fh	DPLLAMD	Display PLL A SDVO/UDI Multiplier/Divsor	R/W
06020h–06023h	DPLLBMD	Display PLL B SDVO/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	—
0620Ch–0620Fh	—	Reserved	—
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

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
07510h–07513h	VF_STR_PSKIP	Debug Starting Primitives Skipped	RO
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	—

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
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
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	—

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
08300h–08303h	MA_DEBUG_1	Message Arbiter Debug Control	R/W
08304h–083FFh	—	Reserved	—
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	—

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
0A800h–0ABFFh	DPALETTE_B	Display Pipe B Palette	R/W
0AC00h–0AFFFh	—	Reserved	—
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

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
3010Ch–3010Fh	OBUF_0V	Overlay Buffer 0 V Pointer	RO
30110h–30113h	OBUF_1U	Overlay Buffer 1 U Pointer	RO
30114h–30117h	OBUF_1V	Overlay Buffer 1 V Pointer	RO
30118h–3011Bh	OSTRIDE	Overlay Stride	RO
3011Ch–3011Fh	YRGB_VPH	Y/RGB Vertical Phase	RO
30120h–30123h	UV_VPH	UV Vertical Phase	RO
30124h–30127h	HORZ_PH	Horizontal Phase	RO
30128h–3012Bh	INIT_PHS	Initial Phase	RO
3012Ch–3012Fh	DWINPOS	Destination Window Position	RO
30130h–30133h	DWINSZ	Destination Window Size	RO
30134h–30137h	SWIDTH	Source Width	RO
30138h–3013Bh	SWIDTHSW	Source Width in Swords	RO
3013Ch–3013Fh	SHEIGHT	Source Height	RO
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

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
3018Ch–3018Fh	OTILEOFF _1Y	Overlay Surface Y 1 Base Address Register	RO
30190h–30193h	OTILEOFF _0U	Overlay Surface U 0 Bae Address Register	RO
30194h–30197h	OTILEOFF _0V	Overlay Surface V 0 Base Address Register	RO
30198h–3019Bh	OTILEOFF _1U	Overlay Surface U 1 Base Address Register	RO
3019Ch–3019Fh	OTILEOFF _1V	Overlay Surface V 1 Base Address Register	RO
301A0h–301A3h	—	Reserved	—
301A4h–301A7h	UVSCALEV	UV Vertical Downscale Integer Register	RO
301A8h–302FFh	—	Reserved	—
30300h–303FFh	Y_VCOEFS	Overlay Y Vertical Filter Coefficients	RO
30368h–303FFh	—	Reserved	—
30400h–305FFh	Y_HCOEFS	Overlay Y Horizontal Filter Coefficient	RO
304ACh–305FFh	—	Reserved	—
30600h–306FFh	UV_VCOEFS	Overlay UV Vertical Filter Coefficients	RO
30668h–306FFh	—	Reserved	—
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	—

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
60050h–60053h	CRCCTRLREDA	Pipe A CRC Red Control	R/W
60054h–60057h	CRCCTRLGREENA	Pipe A CRC Green Control	R/W
60058h–6005Bh	CRCCTRLBLUEA	Pipe A CRC Blue Control	R/W
6005Ch–6005Fh	CRCCTRLRESA	Pipe A CRC Residual Control Register	R/W
60060h–60063h	CRCRESREDA	Pipe A CRC Red Result	RO
60064h–60067h	CRCRESGREENA	Pipe A CRC Green Result	RO
60068h–6006Bh	CRCRESBLUEA	Pipe A CRC Blue Result	RO
6006Ch-6006Fh	CRCRESRESA	Pipe A CRC Residual Result	RO
60070h–60FFFh	—	Reserved	—
Display Pipeline B			
61000h–61003h	HTOTAL_B	Pipe B Horizontal Total	R/W
61004h–61007h	HBLANK_B	Pipe B Horizontal Blank	R/W
61008h–6100Bh	HSYNC_B	Pipe B Horizontal Sync	R/W
6100Ch–6100Fh	VTOTAL_B	Pipe B Vertical Total	R/W
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	ADPA	Analog Display Port A Control	R/W

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
61104h–6110Fh	—	Reserved	—
61110h–61113h	PORT_HOTPLU_EN	Port HotPlug Enable	R/W
61114h–61117h	PORT_HOTPLU_STAT	Port HotPlug Status	R/W
61118h–6113Fh	—	Reserved	—
61140h-61143h	sDVO/HDMIB	Digital Display Port B Control Register	R/W
61144h–6114Fh	—	Reserved	—
61150h-61153h	sDVO/DP	Digital Display Port DFT Register	R/W
61154h–61157h	sDVO/DP	Digital Display Port DFT Register 2	R/W
61158h–6115Fh	—	Reserved	—
61160h-61163h	sDVO/HDMIC	Digital Display Port C l	R/W
61164h–6116Bh	—	Reserved	—
6116Ch–6116Fh	—	Reserved	—
61170h–61173h	VIDEO_DIP_CTL	Video DIP Control	R/W
61174h–61177h	—	Reserved	—
61178h–6117Bh	VIDEO_DIP_DATA	Video Data Island Packet Data	R/W
6117Ch–61177h	—	Reserved	—
LVDS ([DevCL] Only)			
61180h–61183h	LVDS	Digital Display Port Control ([DevCL])	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
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	—

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
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 Audio Registers (62000h–62FFFh)			
62000h–62003h	AUD_CONFIG	Audio Configuration	R/W
62004h–6200Fh	—	Reserved	—
62010h–62013h	AUD_DEBUG	Audio Debug	RO
62014h–6201Fh	—	Reserved	—
62020h–62023h	AUD_VID_DID	Audio Vendor ID / Device ID	RO
62024h–62027h	AUD_RID	Audio Revision ID	RO
62028h–6202Bh	AUD_SUBN_CNT	Audio Subordinate Node Count	RO
6202Ch–6203Fh	—	Reserved	—
62040h–62043h	AUD_FUNC_GRP	Audio Function Group Type	RO
62044h–62047h	AUD_FUNC_SUBN_CNT	Audio Function Subordinate Node Count	RO
62048h–6204Bh	AUD_GRP_CAP	Audio Function Group Capabilities	RO
6204Ch–6204Fh	AUD_PWRST	Audio Power State	RO
62050h–62053h	AUD_SUPPWR	Audio Supported Power State	RO
62054h–62057h	AUD_SID	Audio Root Node Subsystem ID	RO
62058h–6206Fh	—	Reserved	—
62070h–62073h	AUD_OUT_CWCAP	Audio Output Converter Widget Capabilities	RO
62074h–62077h	AUD_OUT_PCMSIZE	Audio PCM Size and Rates	R/W
62078h–6207Bh	AUD_OUT_STR	Audio Stream Formats	R/W
6207Ch–6207Fh	AUD_OUT_DIG_CNVT	Audio Digital Converter	R/W
62080h–62083h	AUD_OUT_CH_STR	Audio Channel ID and Stream ID	RO
62084h–62087h	AUD_OUT_STR_DESC	Audio Stream Descriptor Format	RO
62088h–6209Fh	—	Reserved	—
620A0h–620A3h	AUD_PINW_CAP	Audio Pin Complex Widget Capabilities	RO

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
620A4h–620A7h	AUD_PIN_CAP	Audio Pin Capabilities	RO
620A8h–620ABh	AUD_PINW_CONNLNG	Audio Connection List Length	RO
620ACh–620AFh	AUD_PINW_CONNLST	Audio Connection List Entry	RO
620B0h–620B3h	AUD_PINW_CNTR	Audio Pin Widget Control	RO
620B4h–620B7h	AUD_CNTL_ST	Audio Control State	RO
620B8h–620BBh	AUD_PINW_UNSOLRESP	Audio Unsolicited Response Enable	RO
620BCh–620BFh	AUD_PINW_CONFIG	Audio Configuration Default	RO
620C0h–620D3h	—	Reserved	—
620D4h–620D7h	AUD_HDMIW_STATUS	Audio HDMI Status	R/W
620D8h–6210Bh	—	Reserved	—
6210Ch–62117h	AUD_HDMIW_HDMIEDID	HDMI Data EDID Block	R/W
62118h–62127h	AUD_HDMIW_INFOFR	Audio HDMI Widget Data Island Packet	R/W
62128h–67FFFh	—	Reserved	—
TV Out Control Registers (68000h–6FFFFh)			
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
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

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
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
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

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
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
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

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
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
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	—

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
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			
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	—
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	—

Table 5-2. Memory-Mapped Registers

Address Offset	Symbol	Register Name	Access
Video Sprite A Control : Reserved			
72000h–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	—
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–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	Overscan Color
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.

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

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

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 GEN4 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

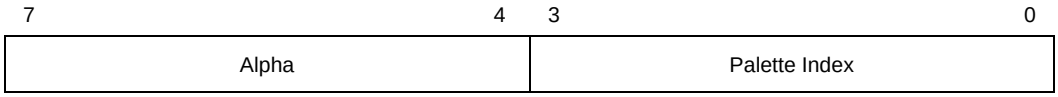
All surface formats containing “I” include an intensity value. When used as a source surface for the sampling engine, the intensity value is replicated to all four channels (R,G,B,A) before being filtered. Intensity surfaces are not supported as destinations.

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

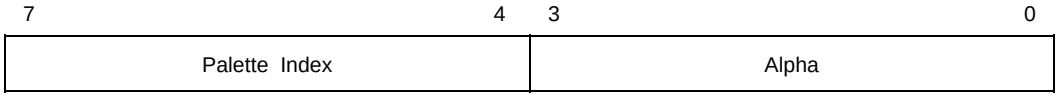
This texel format contains a 4-bit Alpha value (in the high nibble) and a 4-bit Palette Index value (in the low nibble).



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).



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

6.4 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 6-1. FXT1 Format Summary

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

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 6-2. FXT CC_HI Block Encoding

Bit	Description
127:126	Mode = '00'b (CC_HI)
125:121	Color 1 Red
120:116	Color 1 Green
115:111	Color 1 Blue
110:106	Color 0 Red
105:101	Color 0 Green
100:96	Color 0 Blue
95:93	Texel 31 Select
…	…
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 6-3. FXT CC_HI Decoded Colors

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

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

Table 6-4. FXT CC_HI Interpolated Color Table

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

Bit	Description
127:125	Mode = ‘010’b (CC_CHROMA)
124	Unused
123:119	Color 3 Red
118:114	Color 3 Green
113:109	Color 3 Blue
108:104	Color 2 Red
103:99	Color 2 Green
98:94	Color 2 Blue
93:89	Color 1 Red
88:84	Color 1 Green
83:79	Color 1 Blue
78:74	Color 0 Red
73:69	Color 0 Green
68:64	Color 0 Blue
63:62	Texel 31 Select
…	
33:32	Texel 16 Select
31:30	Texel 15 Select
…	
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 6-6. FXT CC_CHROMA Decoded Colors

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

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

Table 6-7. 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 6-8. FXT CC_MIXED Block Encoding

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

Bit	Description
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 6-9. FXT CC_MIXED (Alpha[0]=0) Decoded Colors

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

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

Table 6-10. FXT CC_MIXED Decoded Colors (Alpha[0] = 0)

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 3 [23:17]	Color 3 Red [7:3]	[123:119]
Color 3 [18:16]	Color 3 Red [2:0]	[123:121]
Color 3 [15:11]	Color 3 Green [7:3]	[118:114]
Color 3 [10]	Color 3 Green [2]	[126]
Color 3 [09:08]	Color 3 Green [1:0]	[118:117]
Color 3 [07:03]	Color 3 Blue [7:3]	[113:109]

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
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 figures:

Table 6-11. FXT CC_MIXED Interpolated Color Table (Alpha[0]=0, Texels 0-15)

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

Table 6-12. FXT CC_MIXED Interpolated Color Table (Alpha[0]=0, Texels 16-31)

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

Alpha[0] = 1 Decoding

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

Table 6-13. FXT CC_MIXED (Alpha[0]=0) Decoded Colors

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

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

Table 6-14. FXT CC_MIXED Decoded Colors (Alpha[0] = 1)

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 3 [23:17]	Color 3 Red [7:3]	[123:119]
Color 3 [18:16]	Color 3 Red [2:0]	[123:121]
Color 3 [15:11]	Color 3 Green [7:3]	[118:114]
Color 3 [10]	Color 3 Green [2]	[126]
Color 3 [09:08]	Color 3 Green [1:0]	[118:117]
Color 3 [07:03]	Color 3 Blue [7:3]	[113:109]
Color 3 [02:00]	Color 3 Blue [2:0]	[113:111]
Color 2 [23:19]	Color 2 Red [7:3]	[108:104]
Color 2 [18:16]	Color 2 Red [2:0]	[108:106]
Color 2 [15:11]	Color 2 Green [7:3]	[103:99]
Color 2 [10:08]	Color 2 Green [2:0]	[103:101]
Color 2 [07:03]	Color 2 Blue [7:3]	[98:94]
Color 2 [02:00]	Color 2 Blue [2:0]	[98:96]
Color 1 [23:17]	Color 1 Red [7:3]	[93:89]
Color 1 [18:16]	Color 1 Red [2:0]	[93:91]
Color 1 [15:11]	Color 1 Green [7:3]	[88:84]
Color 1 [10]	Color 1 Green [2]	[125]

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
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 6-15. FXT CC_MIXED Interpolated Color Table (Alpha[0]=1, Texels 0-15)

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

Table 6-16. FXT CC_MIXED Interpolated Color Table (Alpha[0]=1, Texels 16-31)

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

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

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 6-17. FXT CC_ALPHA Block Encoding

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

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 6-18. FXT CC_ALPHA Decoded Colors

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

LERP = 0 Decoding

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

Table 6-19. FXT CC_ALPHA Interpolated Color Table (LERP=0)

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

LERP = 1 Decoding

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

Table 6-20. FXT CC_ALPHA Interpolated Color Table (LERP=1, Texels 0-15)

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

Table 6-21. FXT CC_ALPHA Interpolated Color Table (LERP=1, Texels 16-31)

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

6.4.2 BC Texture Formats

The hardware supports three "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 formats strictly use four colors, as the alpha values are obtained from the alpha block (the DX7 Direct3D reference rasterizer had a known bug that erroneously allowed 3-color BC2-3 color blocks).

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)

6.5 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.

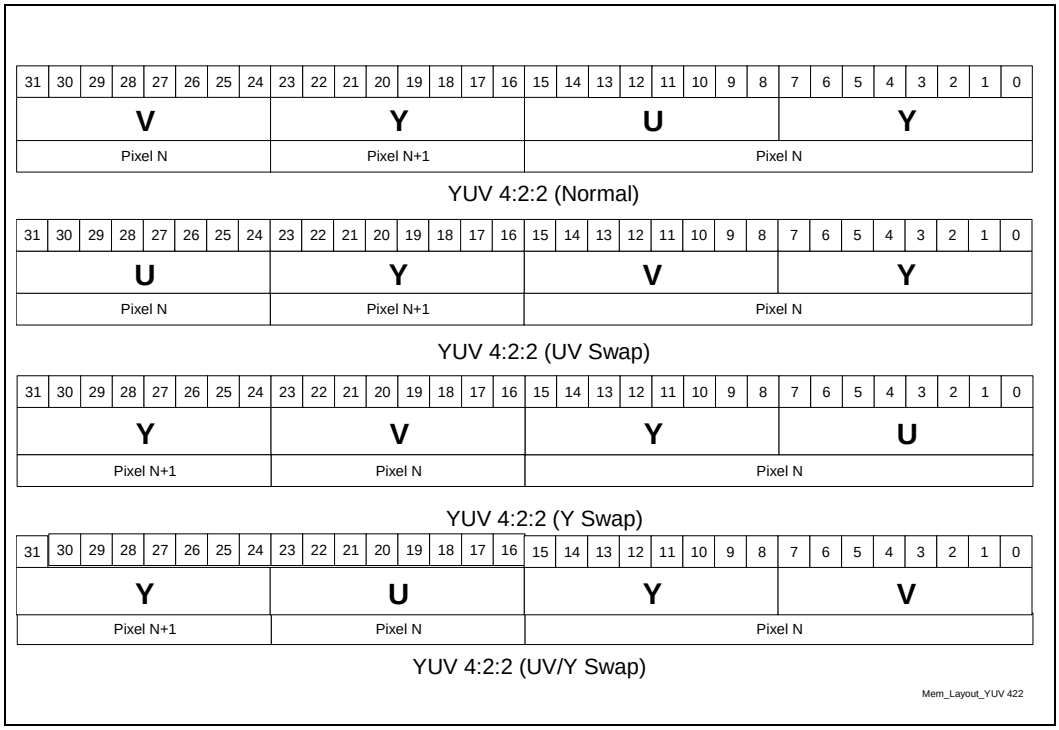
The following four YUV 4:2:2 surface formats are supported, listed with alternate names:

- 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



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.

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

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

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

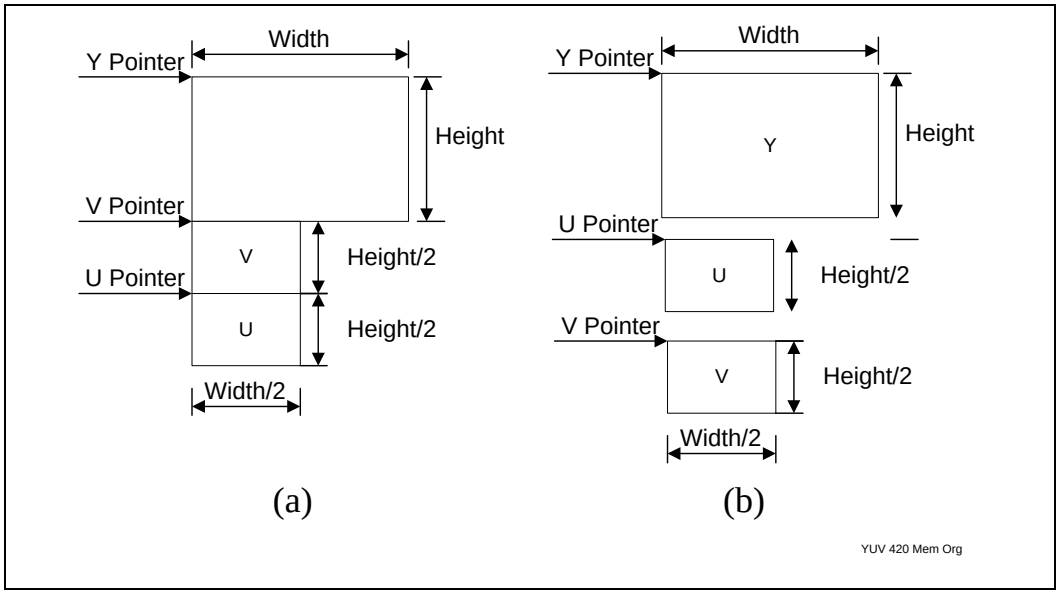
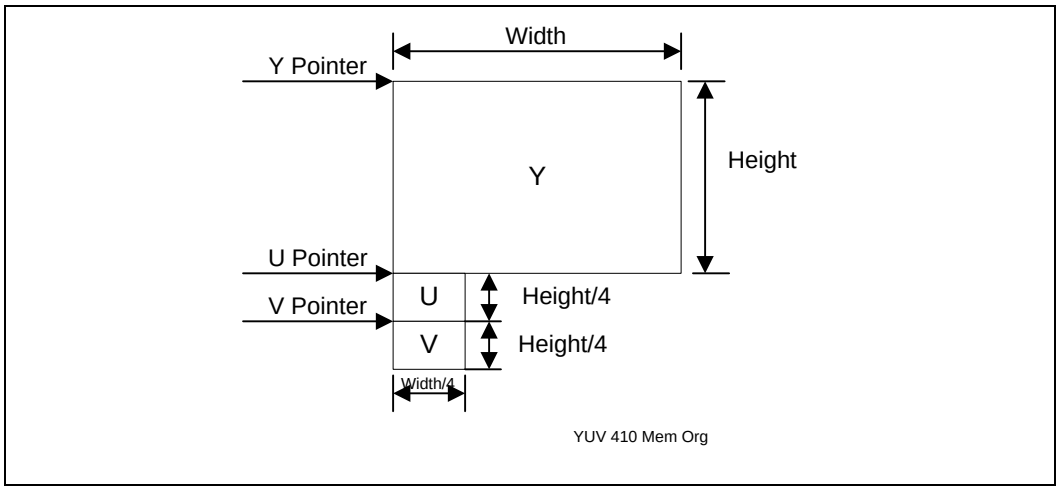


Figure 6-3. YUV 4:2:0 Format Memory Organization

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 6-22. Depth Buffer Formats

DepthBufferFormat / DepthComponent	bpp	Description
D32_FLOAT_S8X24_UINT	64	32-bit floating point Z depth value in first DWord, 8-bit stencil in 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.

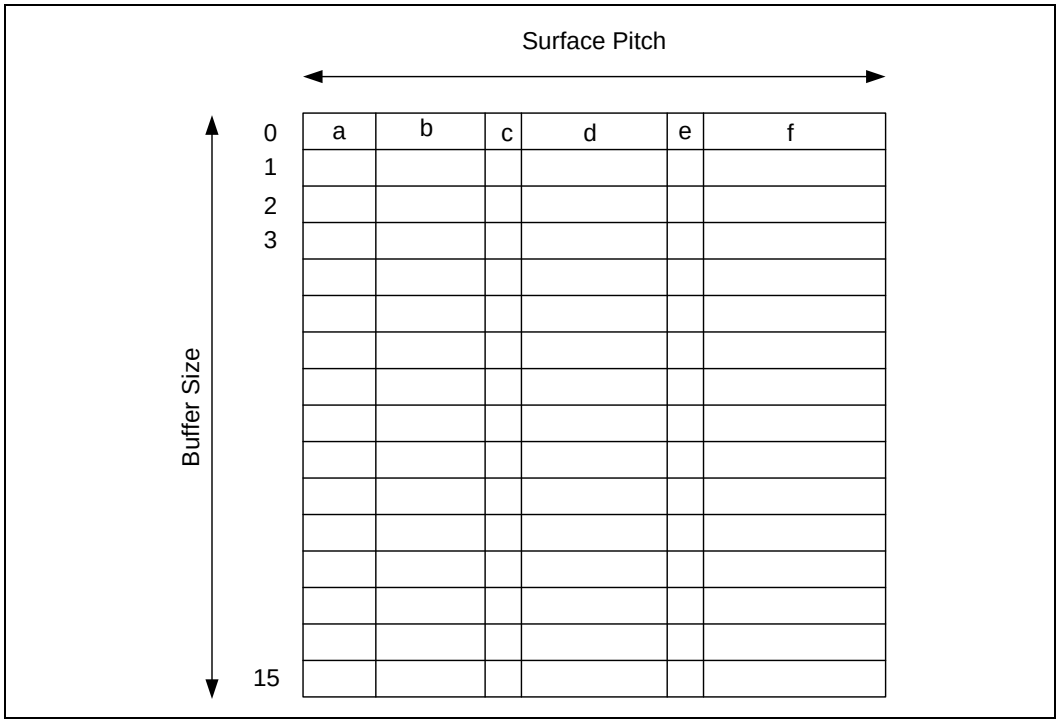
6.17 Surface Layout

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:

$$W_L = \left(\left(width >> L \right) > 0 \,?\, width >> L : 1 \right)$$
$$H_L = \left(\left(height >> L \right) > 0 \,?\, height >> L : 1 \right)$$

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

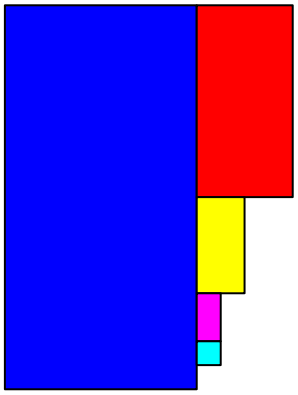
To account for the cache line alignment required, we define *i* and *j* as the width and height, respectively, of an *alignment unit*. This alignment unit is defined below. We then define lower-case w_L and h_L as the padded width and height of LOD "*L*" as follows:

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

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

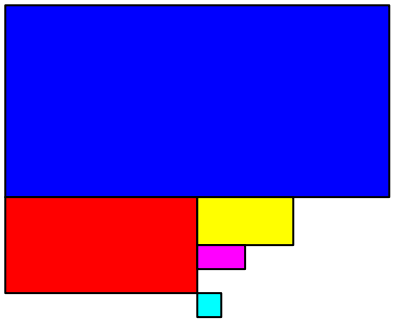
for *MIPLAYOUT_RIGHT* maps:

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



for *MIPLAYOUT_BELOW* maps:

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



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 6-23. 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 BC1-3 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:

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

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

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

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

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

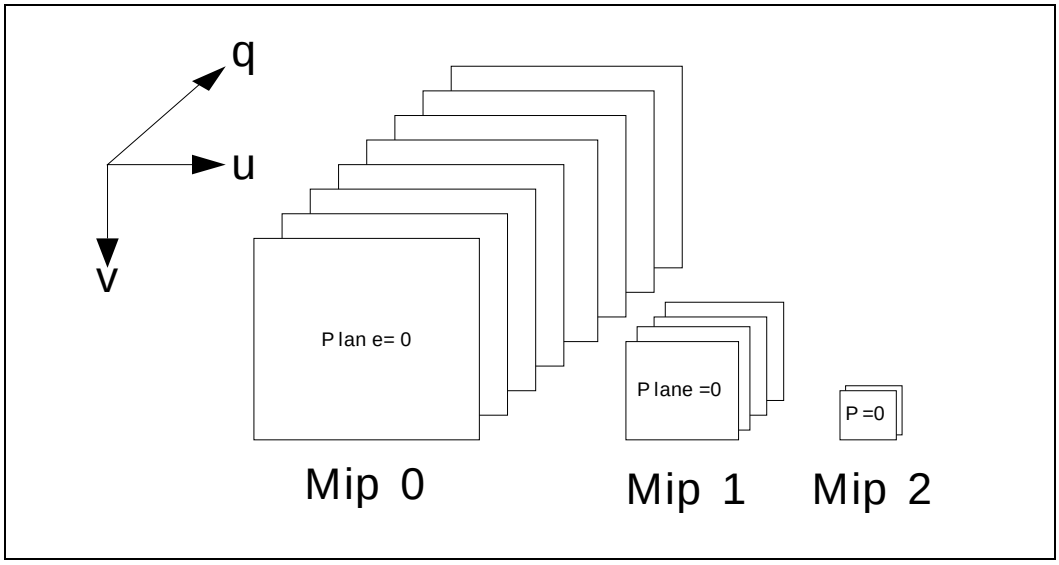
6.17.4.2 Restrictions

- 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 as shown in Table 6-23.

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

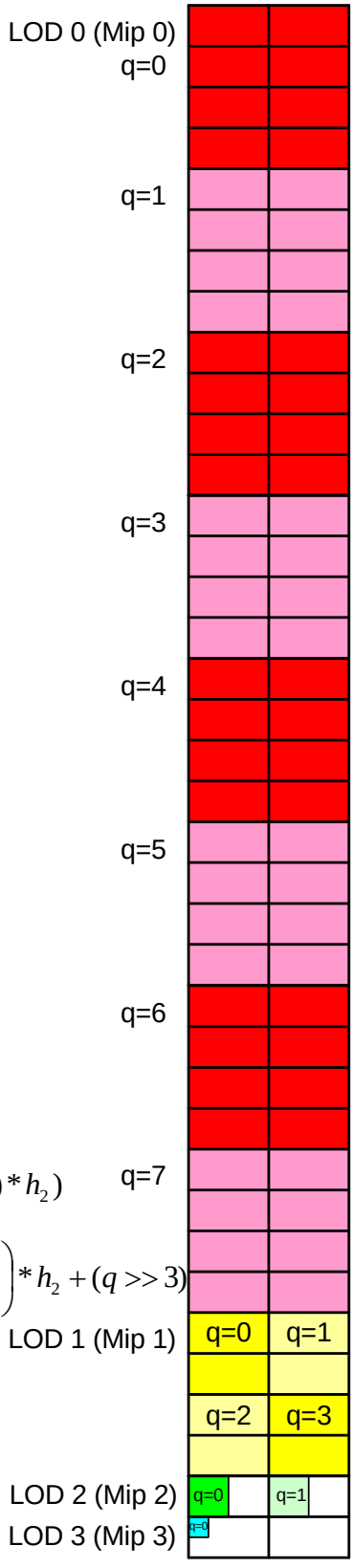
Note that it is not necessary to cache-line align in the "depth" dimension (i.e. lower case "*d*").

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

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

...

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.

6.19.2 Register/State Context

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

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-29h	31:0	**Reserved**
2Ah	31:0	**PS_DEPTH_COUNT Lower Address**
2Bh	31:0	**PS_DEPTH_COUNT Lower Data**

DWord	Bits	State Field
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)**
	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)

DWord	Bits	State Field
	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**
	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	**DWord 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	ength **(Excludes DWord 0,1) = 2**

DWord	Bits	State Field
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)
	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

DWord	Bits	State Field
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
	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

DWord	Bits	State Field
	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**
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**

DWord	Bits	State Field
	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**
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]**
F5h	31:0	**Reserved**

DWord	Bits	State Field
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)
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

DWord	Bits	State Field
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**
	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**

DWord	Bits	State Field
	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. Table 6-25 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 6-24) 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 6-24. 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 6-25. 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
STATE_BASE_ADDRESS	
General State Base Address	0

Instruction / Field	Value
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
3DSTATE_CONSTANT_COLOR	
Blend Constant Color Red	1.0
Blend Constant Color Blue	1.0

Instruction / Field	Value
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
Buffer Pitch	0
Buffer Starting Address	0
Max Index	0
… values repeated for all 17 Vertex Buffers	…

Instruction / Field	Value
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_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

6.19.3 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 Gen4 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	0000000000000000Ch	RO; R/W; R/W/L;
I/O Base Address	IOBAR	20	23	00000001h	RO; R/W;

Register Name	Register Symbol	Register Start	Register End	Default Value	Access
Subsystem Vendor Identification	SVID2	2C	2D	0000h	R/WO;
Subsystem Identification	SID2	2E	2F	0000h	R/WO;
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	MDEVENdev0F0	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	0000000000000000h	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;

Register Name	Register Symbol	Register Start	Register End	Default Value	Access
GMBUS frequency binary encoding	GMBUSFREQ	CC	CD	0000h	R/W; RO;
Power Management Capabilities ID	PMCAPID	D0	D1	0001h	RO;
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

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.

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.
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 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	0000000000b	**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

As long as there is the potential that DVO port legacy drivers exist which expect this register at this address, Dev#2F0address E0h-E1h must be reserved for this register.

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	000000000 0000004h	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	MDEVENdev0 F0	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

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**

7.3.21 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.

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**

7.3.23 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**

7.3.24 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**

7.3.25 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. 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	**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.
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 Gen4 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

Gen4 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:

1. *Flush the Gfx Pipeline*
2. *Flush the Chipset write buffers using the flush (GFX_FLSH_CTL) register*
3. *Update Global GTT using physical address/Update the Global GTT base register*
4. *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.

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 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 **Page Table Enable** field is specified as ENABLED. **Programming Notes** — **Project** 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 — DevCL
11:8	**Reserved** Project: All Format: MBZ

PGTBL_CTL—Page Table Control Register

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

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
24	**LC_INVALID_GTT_PTE** Project: All Default Value: 0h Format: Flag Invalid Render Cache GTT entry **Errata** **Description** **Project** GEN4016 This bit will never be set. All
23	**ISC_INVALID_GTT_PTE** Project: All Format: Flag Invalid Instruction/State Cache GTT entry

PGTBL_ER—Page Table Error Register

22	**ROC_INVALID_GTT_PTE** Project: All Format: Flag Reserved since there is no ROC
21	**CS_VertexData_INVALID_GTT_PTE** Project: All Format: Flag Invalid GTT Entry during Vertex Fetch
20	**CS_Command_INVALID_GTT_PTE** Project: All Format: Flag Invalid GTT Entry during Command Fetch
19	**CS_INVALID_GTT** Project: All Format: Flag
18	**CRSR _INVALID_GTT_PTE** Project: All Format: Flag Invalid GTT Entry during Cursor Fetch
17	**Reserved** Project: All Format: MBZ
16	**OVRL_INVALID_GTT_PTE** Project: All Default Value: 0h Format: Flag Invalid GTT Entry during Overlay Fetch **Errata** **Description** **Project** BWT010 Invalid GTT Entry during Overlay Fetch is ignored. DevBW-A, DevBW-B
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

PGTBL_ER—Page Table Error Register	
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 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.

8.2.1.4 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).

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

8.2.2.1 PGTBL_CTL2— Per Process Page Table Control Register

PGTBL_CTL2— Per Process Page Table Control Register	
Register Type:	MMIO
Address Offset:	20C4h
Project:	All
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
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.

PGTBL_CTL2— Per Process Page Table Control Register

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

8.2.2.2 PGTBL_STR2—Page Table Steer Register (Per Process)

PGTBL_STR2—Page Table Steer Register (Per Process)

Register Type: MMIO
Address Offset: 20C8h
Project: All
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.

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

PGTBL_STR2—Page Table Steer Register (Per Process)

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

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

8.2.3 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.

8.2.3.1 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.

8.2.3.2 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.

8.2.3.3 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.

8.2.3.4 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.

8.2.3.5 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.

8.2.3.6 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						
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

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

5	**Indirect State Pointer Force Restore** Determines whether to use pending indirect state counter to restore data to memory, or restore indirect data

Value	Name	Description	Project
0h	Use Pending	Use the pending indirect state counter to restore data to memory	All
1h	Don't Use Pending	Don't use pending indirect state counter to restore data to memory. Always restore indirect data	All

4:0	**User Defined Condition Codes** The software may signal a Stream Semaphore by setting the Mask bit and Signal Bit together to match the bit field specified in a WAIT_FOR_EVENT (Semaphore).

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.

Bit	Description
31:21	**Wrap Count** Project: All Default Value: 0h Format: U11 count of ring buffer wraps This field is incremented by 1 whenever the **Head Offset** wraps from the end of the buffer back to the start (i.e., whenever it wraps back to 0). Appending this field to the **Head Offset** field effectively creates a virtual 4GB Head "Pointer" which can be used as a tag associated with instructions placed in a ring buffer. The Wrap Count itself will wrap to 0 upon overflow. The Wrap Count will get cleared as a result of writes of the Starting Address field.
20:2	**Head Offset** Project: All Format: U19 DWord 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. **Programming Notes** **Project** A RB can be enabled empty or containing some number of valid instructions. All Head Offset is cleared as a result of writes of the Starting Address field. All
1	**Reserved** Project: All Format: MBZ
0	**Wait for Condition Indicator** Project: All Format: Enabled This is a read only value used to indicate whether or not the command streamer is currently waiting for a conditional code to be cleared from 0x2028

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
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	Report every 16 pages (64KB)	All
	MI_AUTOREPORT_4KB	When the **Per-Process Virtual Address Space and Run List Enable** bit is set, the ring buffer reports every 4KB	
2h	Reserved	Reserved	All
3h	MI_AUTOREPORT_128KB	Report every 32 pages (128KB)	All

RING_BUFFER_CONTROL	
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).

Bit	Description
31:4	**Reserved** Project: All Format: MBZ
3	**Batch Buffer Error** Project: All Format: Flag 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	**Ring ID** Project: All Default Value: 0h Format: U3 This field indicates which ring buffer is associated with the faulting

Value	Name	Description	Project
0	Ring Buffer 0	Ring Buffer 0	All
1-7	Reserved	Reserved	All

8.6.3 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.

8.6.4 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**

INSTDONE—Instruction Stream Interface Done Register (Debug)	
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
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

8.6.5 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.

8.6.6 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

8.6.7 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

8.6.8 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**

INSTDONE_1 — Additional Instruction Stream Interface Done (Debug)	
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**
3	**cl_cs_done**
2	**gs_cs_done**
1	**vs0_cs_done**
0	**vf_cs_done**

8.6.9 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.

8.7 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		
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 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	Reserved.
24	Reserved.
23	Reserved.
22	Reserved.
21	Reserved.
20	Reserved.
19	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	**Display Port Interrupt**: This status bit is set when a port hotplug/unplug event has been detected. The specific trigger of this interrupt can be read in the port hotplug status register.
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**

Bit	Description
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 Gen4 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	Reserved. MBZ	
24	Reserved. MBZ	
23	Reserved. MBZ	
22	Reserved. MBZ	
21	Reserved. MBZ	
20	Reserved. MBZ	
19	Reserved. MBZ	
18	PIPE_CONTROL packet - Notify Enable	0
17	Display Port Interrupt	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	Overlay Flip Pending	Set when flip is pending
8	Display Plane C 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

8.8.1 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

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** Project: All Security: None Address: PhysicalAddress[31:12] Surface Type: U32 Range 0..2^32-1 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.
12:8	**Reserved** Project: All Format: MBZ
7:4	**Physical Start Address Extension** Project: All Security: None Address: PhysicalAddress[35:32] This field specifies Bits 35:32 of the starting physical address.
3:0	**Reserved** Project: All Format: MBZ

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

DWord Offset	Description
0	**Interrupt Status Register Storage:** The content of the ISR register is written to this location whenever an “unmasked” bit of the ISR (as determined by the HWSTAM register) changes state.
3:1	**Reserved.** Must not be used.
4	**Ring Head Pointer Storage:** The contents of the Ring Buffer Head Pointer register (register DWord 1) are written to this location either as result of an MI_REPORT_HEAD instruction or as the result of an “automatic report” (see RINGBUF registers).
Fh:5h	**Reserved.** Must not be used.
10h-1Bh	**Reserved.** Must not be used.
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.

8.8.2 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.

Bit	Description
31:12	**Power Context Address** Project: DevCL Default Value: 0h Address: GraphicsAddress[31:12] 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.
11:5	**Reserved** Project: DevCL Format: MBZ

PWRCTXA — Power Context Register Address	
4:1	**Power Context Size** Project: DevCL Default Value: 0h Format: U4 Field specifies the size of the power context allocated by the software. The size is in terms of 4K pages This field is ReadOnly **Value** / **Name** / **Description** / **Project** 001-111 / Reserved / Reserved / All 000 / 4KB / / DevCL
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

8.8.3 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

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.

Bit	Description
31:13	**Reserved** Project: All Format: MB1

Hardware Status Mask Register

12	**Sync Status** 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 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). All
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

Hardware Status Mask Register	
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
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
7:0	**Reserved** Project: All Format: MB1

8.8.4 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.

Bit	Description
31:0	**Interrupt Enable Bits** Project: All Default Value: 0h Format: Array of Enables — refer to Table 8-1 in Interrupt Control Register section for bit definitions The bits in this register enable a CPU interrupt to be generated whenever the corresponding bit in the Interrupt Identity Register becomes set.

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

8.8.5 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.

Bit	Description
31:0	**Interrupt Identity Bits** Project: All Default Value: 0h Format: Array of unmasked — Persistent interrupt bits (refer to Table 9-1 in Interrupt Control Register section for bit definitions) 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). **Value** / **Name** / **Description** / **Project** 1h / Interrupt Condition Detected / Interrupt Condition Detected (may or may not have actually generated a CPU interrupt) / All **Programming Notes** 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.

8.8.6 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.

Bit	Description
31:0	**Interrupt Mask Bits** Project: All Default Value: FFFE DFFFh Format: Array of interrupt mask bits — Refer to Table 9-1 in Interrupt Control Register section for bit definitions This field contains a bit mask which selects which interrupt bits (from the ISR) are reported in the IIR.

Value	Name	Description	Project
0h	Not Masked	Will be reported in the IIR	All
1h	Masked	Will not be reported in the IIR	All

8.8.7 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 9-1 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

8.9 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	Reserved: MBZ
9	Reserved.
8	Reserved.
7:6	Reserved: MBZ
5	Reserved.
4	**Page Table Error**: This bit is set when a Graphics Memory Mapping Error is detected. The cause of the error is indicated (to some extent) in the PGTBL_ER register. Note: This error indications cannot be cleared except by reset (i.e., it is a fatal error). 1 = Page table error
3	Reserved.
2	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.

8.9.1 EIR — Error Identity Register

EIR — Error Identity Register

Register Type: MMIO
Address Offset: 20B0h
Project: All
Default Value: 0000 0000h
Access: R/WC
Size (in bits): 32

The EIR register contains the persistent values of Hardware-Detected Error Condition bits. Any bit set in this register will cause the Master Error bit in the ISR to be set. The EIR register is also used by software to clear detected errors (by writing a '1' to the appropriate bit(s)).

Bit	Description
31:16	**Reserved** Project: All Format: MBZ
15:0	**Error Identity Bits** Project: All Default Value: 0h Format: Array of Error condition bits See Table 9-5. Hardware-Detected Error Bits This register contains the persistent values of ESR error status bits that are unmasked via the EMR register. (See Table 8-2. Hardware-Detected Error Bits). The logical OR of all (defined) bits in this register is reported in the Master Error bit of the Interrupt Status Register. In order to clear an error condition, software must first clear the error by writing a '1' to the appropriate bit(s) in this field. If required, software should then proceed to clear the Master Error bit of the IIR. **Value** **Name** **Description** **Project** 1h Error occurred Error occurred All **Programming Notes** **Project** Writing a '1' to a set bit will cause that error condition to be cleared. However, the Page Table Error bit (Bit 4) cannot be cleared except by reset (i.e., it is a fatal error). All

8.9.2 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.

Bit	Description
31:16	**Reserved** Project: All Format: MBZ
15:0	**Error Mask Bits** Project: All Default Value: FFFF FFDFh Format: Array of error condition mask bits See Table 9-5. Hardware-Detected Error Bits This register contains a bit mask that selects which error condition bits (from the ESR) are reported in the EIR.

Value	Name	Description	Project
0h	Not Masked	Will be reported in the EIR	All
1h	Masked	Will not be reported in the EIR	All

8.9.3 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** Project: All Default Value: 0h Format: Array of error condition bits See Table 9-5. Hardware-Detected Error Bits This register contains the non-persistent values of all hardware-detected error condition bits. **Value** **Name** **Description** **Project** 1h Error Condition Detected Error Condition detected All

8.10 Register Definitions for Context Save

8.10.1 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

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	Reserved. MBZ
9:8	**Reserved** Project: All Format: MBZ
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

INSTPM—Instruction Parser Mode Register

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
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

8.10.2 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.
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

Cache_Mode_0— Cache Mode Register 0

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

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

Cache_Mode_0— Cache Mode Register 0

8	**Reserved** Project: All Format: MBZ
7:6	**Sampler L2 Request Arbitration** Project: All Default Value: 0h Format: U2 **Value** **Name** **Description** **Project** 00 Round Robin All 01 Fetch are Highest Priority All 10 Constants are Highest Priority All 11 Reserved All
5	**Reserved.** This bit must be 0. Note that it defaults to 1.
4:3	**Reserved** Project: All Format: MBZ
2	**Reserved** Project: All Format: MBZ
1	**Reserved** Project: All Format: MBZ
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

8.10.3 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.

Bit	Description
31:16	**Mask Bits for 15:0** Format: Mask[15:0] Must be set to modify corresponding data bit. Reads to this field returns zero.
15:13	**Reserved** Project: All Format: MBZ
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

Cache_Mode_1— Cache Mode Register 1

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

10	**Instruction Level 1 Cache and In-Flight Queue Disable** Project: All Default Value: 0h Format: Disable

Value	Name	Description	Project
0h		Cache is enabled.	All
1h		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)	All

9	**Instruction and State Level 2 Cache Fill Buffers Disable** Project: All Default Value: 0h Format: Disable

Value	Name	Description	Project
0h		Fill Buffers are enabled.	All
1h		Fill Buffers are disabled. (DEBUG ONLY)	All

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

6:5	**Reserved**	Project:	All	Format:	MBZ
4	**Reserved**	Project:	All	Format:	MBZ

Cache_Mode_1— Cache Mode Register 1

3	**Reserved** Project: DevCL Format: MBZ
3	**A-step bug fix bit for rcc allocation** Project: DevBW-A, DevBW-B Default Value: 0h Format: U1 This bit should always be set for proper operation on BW-A,B

Value	Name	Description	Project
0h		This bug fix is disabled.	DevBW-A, DevBW-B
1h		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.	DevBW-A, DevBW-B

2	**Reserved** Project: All Format: MBZ
1	**Instruction and State Level 2 Cache Disable** Project: All Default Value: 0h Format: Disable

Value	Name	Description	Project
0h		Cache is enabled.	All
1h		Cache is disabled and all accesses to this cache are treated as misses. (DEBUG ONLY)	All

0	**Instruction Level 1 Cache Disable** Project: All Default Value: 0h Format: Disable

Value	Name	Description	Project
0h		Cache is enabled.	All
1h		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)	All

8.10.4 BB_ADDR—Batch Buffer Head Pointer Register

BB_ADDR—Batch Buffer Head Pointer Register

Register Type: MMIO
Address Offset: 2140h
Project: All
Default Value: 0000 0000 0000 0000h
Access: RO
Size (in bits): 64

This register contains the current DWord Graphics Memory Address of the last-initiated batch buffer.

Bit	Description
63:32	**Reserved** Project: All Format: MBZ
31:2	**Batch Buffer Head Pointer** Project: All Format: GraphicsAddress[31:2] This field specifies the DWord-aligned Graphics Memory Address where the last initiated Batch Buffer is currently fetching commands. If no batch buffer is currently active, the Valid bit will be 0 and this field will be meaningless.
1	**Reserved** Project: All Format: MBZ
0	**Valid** Project: All Default Value: 0h Format: U1 **Value** **Name** **Description** **Project** 0h Invalid Batch buffer Invalid All 1h Valid Batch buffer Valid All

8.10.5 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.

Bit	Description
31:6	**Reserved** Project: All Format: MBZ
5	**Buffer Security Indicator** Project: All Default Value: 0h Format: MI_BufferSecurityType If set, this batch buffer is non-secure and cannot execute privileged commands nor access privileged (GGTT) memory. It will be accessed via the PPGTT. If clear, this batch buffer is secure and will be accessed via the GGTT. Note: This field reflects the effective security level and may not be the same as the Buffer Security Indicator written using MI_BATCH_BUFFER_START. **Value** / **Name** / **Description** / **Project** 0h / MIBUFFER_SECURE / Located in GGTT memory / All 1h / MIBUFFER_NONSECURE / Located in PPGTT memory / All
4:0	**Reserved** Project: All Format: MBZ

8.10.6 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.

Bit	Description
31:2	**Reserved** Project: All Format: MBZ
1	**Extended Context Enable** Project: All Default Value: 0h Format: Enable 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. **Value** / **Name** / **Description** / **Project** 0h / Disable / The current context does not include extended context / All 1h / Enable / The current context does include extended context. / All
0	**Render Context Restore Inhibit** Project: All Format: U1 This is not a true register bit. This bit should be set in the context image of a ring context that is being submitted for the first time. Setting this bit will inhibit the restoring of render context (including extended context if applicable) so that restoring of an uninitialized render context can be prevented. This bit will always be set on a context save (since the render context cannot be uninitialized on context save – it will always contain at least default values.)

8.11 Logical Context Support

8.11.1 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 Enable** Project: All Format: Enable If set, the extended state identified in the Logical Context Data section of the Memory Data Formats chapter, is saved as part of switching away from this logical context.

CCID—Current Context ID Register

2	**Extended State Restore Enable** Project: All Format: Enable If set, the extended state identified in the Logical Context Data section of the Memory Data Formats chapter, was loaded (or restored) as part of switching to this logical context.
1	**Reserved** Project: All Format: MBZ
0	**Valid** Project: All Default Value: 0h Format: U1

Value	Name	Description	Project
0h	Invalid	The other fields of this register are invalid. A switch away from the context will not invoke a context save operation.	All
1h	Valid	The other fields of this register are valid, and a switch from the context will invoke the normal context save/restore operations.	All

8.11.2 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.

8.11.3 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
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.

8.12 Arbitration Control, and Scratch Bits

8.12.1 MI_DISPLAY_POWER_DOWN—Display Power Down ([DevCL] Only)

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

8.12.2 MI_ARB_STATE—Memory Interface Arbitration State Register

MI_ARB_STATE—Memory Interface Arbitration State Register

Register Type: MMIO
Address Offset: 20E4h
Project: All
Default Value: 0000 0040h
Access: R/W
Size (in bits): 32

The MI_ARB_STATE register contains state information that controls arbitration aspects of the Memory Interface function.

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	**Render/Sampler TLB Request Priority** Project: All Default Value: 0h Format: U1 **Value** **Name** **Description** **Project** 0h 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. All 1h TLBs are at the lowest priority (above FBC) with Sampler TLB fetches higher than render. All
14:9	**Reserved** Project: All Format: MBZ Read/Write (SW must maintain setting)

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

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

MI_ARB_STATE—Memory Interface Arbitration State Register

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)

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

8.12.3 MI_RDRET_STATE—Memory Interface Read Return State Register

MI_RDRET_STATE—Memory Interface Read Return State Register	
Register Type:	MMIO
Address Offset:	20FCh [DevCL] 20E0h [DevBW]
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

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.

Bit	Description
31:16	**Mask Bits** Format: Mask[15:0] Must be set to a 1 to allow modification of corresponding bit in Bits 15:0. (All implemented bits)
15	**HVM Enable Bit** Project: All Format: Enable 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.
14:9	**Reserved** Project: All Format: MBZ
8	**Vertex Fetch Cache select** Project: All Default Value: 0h Format: U1 This bit determines the read return for Vertex Fetch Reads **Value** **Name** **Description** **Project** 0h Return Data from memory All 1h Return data from a random data generator on-chip All
7	**Reserved** Project: All Format: MBZ Was Read Only Cache select

MI_RDRET_STATE—Memory Interface Read Return State Register

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

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

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

8.12.4 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.

Bit	Description
31:16	**Masks** Format: Mask[15:0] A “1” in a bit in this field allows the modification of the corresponding bit in Bits 15:0
15	**Reserved** Project: All Format: MBZ Read/Write
14	**Reserved** Project: All Format: MBZ Read/Write
13	**Flush Performance mode** Project: DevCL Default Value: 0h Format: U1 **Value** **Name** **Description** **Project** 0h run fast restore No NonPipelined SV flush. DevCL 1h run slow legacy restore With NonPipelined SV flush. DevCL
13	**Reserved** Project: DevBW Format: MBZ Read/Write
12	**Reserved** Project: All Format: MBZ
11	**Invalidate UHPTR enable** Project: All Format: Enable If bit set H/W clears the valid bit of UHPTR (2134h, bit 0) when current active head pointer is equal to UHPTR.
10	**Power of 2 Fences Enable** Project: All Format: Enable 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.

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
7	**Vertex Shader Cache Mode** Project: All Default Value: 0h Format: U1 **Value** **Name** **Description** **Project** 0h Non-LRA Non-LRA mode of allocation. Vertex shader cache is allocated on the basis of the reference count of individual vertices All 1h LRA LRA mode of allocation. Used for validation purposes. All

MI_MODE — Mode Register for Software Interface

6 — Vertex Shader Timer Dispatch Enable

Project: All
Default Value: 0h
Format: Enable

Value	Name	Description	Project
0h	Disable	Disable the timer for dispatch of single vertices from the vertex shader. Vertex shader will try to collect 2 vertices before a dispatch	All
1h	Enable	Enable the timer for dispatch of single vertices. Dispatch a single vertex shader thread after the timer expires.	All

Programming Notes — **Project**

To avoid deadlock conditions in hardware this bit needs to be set for normal operation. — All

5 — 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

4 — Reserved: MBZ

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 — Reserved Project: All Format: MBZ

Read/Write

MI_MODE — Mode Register for Software Interface					
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.				

8.12.5 ECOSKPD—ECO Scratch Pad (DEBUG)

ECOSKPD—ECO Scratch Pad (DEBUG)

Register Type: MMIO
Address Offset: 21D0h
Project: All
Default Value: 00000307h
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 implemented bits)
15	**Reserved** Project: All Format: MBZ
14	**Vertex Shader Dual dispatch disable** Project: DevBW-E Security: None Default Value: 0h Enable the dual dispatch Mask: MMIO(0x21D0)#30 **Value** **Name** **Description** **Project** 0h Enable HW implements the fix for the enhanced dual dispatch. Dual dispatch is triggered only when the top entry in the tracking FIFO is a DevBW-E 1h Disable Disable the HW fix for the enhanced dual dispatch. Vertex shader will be dispatched as a single vertex everytime the tracking FIF becomes full. DevBW-E
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

ECOSKPD—ECO Scratch Pad (DEBUG)

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	DevBW-E

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.

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

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

ECOSKPD—ECO Scratch Pad (DEBUG)	
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	**Reserved** Project: All Format: MBZ
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
3	**WIZunit Scratch Space ECO** Project: DevBW-C+ Default Value: 0h DefaultVaueDesc Format: U1 FormatDesc Enable ECO: Max scratch space (indicated by **Per Thread Scratch Space** set to 11) is 256KB. 256KB scratch base must be 8M aligned. Disable ECO: Max scratch space is 12KB.
2:0	**Reserved** Project: All Format: MBZ

8.13 Debug Registers

These registers are used to reflect internal hardware state. The intention is to be used for silicon debug

8.13.1 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”

8.13.2 CSFLFLAG — Flush FLAG (Debug)

CSFLFLAG — Flush FLAG (Debug)					
Register Type:	MMIO				
Address Offset:	2204h				
Project:	All				
Default Value:	0000 0000h				
Access:	R/W				
Size (in bits):	32				

Bit	Description
31:17	**Reserved: 0x0** Project: All Format: MBZ
16:9	Project: All Format: U8 csprsrallflsh& csctxlcflsh& csynclcflush &fi_write & fi_depth & fi_timestamp & fi_iscflush & fi_globalcnt_rst
8	**cs_media_select** Project: All Format: U1
7:0	Project: All Format: U8 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

8.13.3 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]

8.13.4 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

8.13.5 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

8.13.6 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.

8.13.7 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.

8.13.8 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

8.14 Software Visible Counter Registers

These registers keep continuous count of time and pixels passing the depth test. 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.

8.14.1 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

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.

8.14.2 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 8.20.

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

Bit	Description
63:0	**TIMESTAMP** Project: All Format: U64 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 **hclk** domain it is not reset by a graphics reset alone.

8.15 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

8.16 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.

8.16.1.1 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

8.16.1.2 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.

8.16.1.3 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

8.16.1.4 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

8.16.1.5 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 – 010h)	These locations can be used for general purpose via the MI_STORE_DATA_INDEX or MI_STORE_DATA_IMM instructions.

8.16.1.6 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.

8.16.1.7 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. 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

8.16.1.8 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

8.16.1.9 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.

8.16.1.10 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

8.16.1.11 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.

8.16.1.12 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

8.16.1.13 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.

8.16.1.14 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.

8.17 Software Control Bit Definitions

Registers in the range 22XX are not protected from the load register immediate instruction if the command is executed in the non-secure batch buffer.

8.18 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]

8.18.1 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

8.18.2 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.**

Bit	Description
31:12	**Compressed Frame Line Length 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 Line Length Buffer. **Programming Notes** 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 **Enable Frame Buffer Compression** has been cleared.)
11:0	**Reserved** Project: DevCL Format: MBZ

8.18.3 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.

Bit	Description
31	**Enable Frame Buffer Compression** Project: DevCL Default Value: 0h Format: Enable This bit is used to globally enable or disable the RLE-FBC function (compression and decompression) at the next VBlank start. **Value / Name / Description / Project** 0h / Disable / Disable frame buffer compression. / DevCL 1h / Enable / Enable frame buffer compression. / DevCL
30	**Mode Select** Project: DevCL Default Value: 0h Format: U1 **Value / Name / Description / Project** 0h / Single Pass / Single Pass mode / DevCL 1h / Periodic Pass / Periodic mode / DevCL
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.

FBC_CONTROL — Frame Buffer Compression Control Register	
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.

8.18.4 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').

8.18.5 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.
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.

FBC_STATUS — Frame Buffer Compression Status Register

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.

8.18.6 FBC_CONTROL2— Frame Buffer Compression 2nd Control Register

FBC_CONTROL2— Frame Buffer Compression 2nd Control Register

Register Type: MMIO
Address Offset: 3214h
Project: DevCL
Default Value: 0000 0000h
Access: R/W
Size (in bits): 32

This register is used to control the operation of RLE-FBC.

Bit	Description
31:3	**Reserved** Project: DevCL Format: MBZ
4	**Double Buffer FBC Fence and Fence_DisplayY Offset Register Fields** Project: DevCL Default Value: 0h Format: Disable **Value** **Name** **Description** **Project** 0h Double buffer DevCL 1h Don't double buffer DevCL
3:2	**FBC C3 Mode** Project: DevCL Default Value: 0h Format: U2 **Value** **Name** **Description** **Project** 00 FBC IDLENESS is not looked at in order to enter Self Refresh DevCL 01 FBC IDLENESS is looked at in order to enter Self Refresh DevCL 10 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. DevCL 11 Reserved Reserved DevCL

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

8.18.7 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

8.18.8 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

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.

8.18.9 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

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
	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

FBC_TAG — Frame Buffer Compression TAG Interface (DEBUG)

	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
	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	Modified	At least one of the associated lines was modified since the last compression pass started.	All
01	Uncompressed	The associated lines are uncompressed and are candidate for compression in the next pass	All
10	Uncompressible	The associated lines are uncompressible and are not candidate for compression in the next pass.	All
11	Compressed	The associated lines are compressed	All

8.19 Fence Registers

8.19.1 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

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

DWord	Bit	Description
0..15	63:44	**Fence Upper Bound** Project: All Address: GraphicsAddress[31:12] Bits 31:12 of the ending Graphics Address of the fence region. Fence regions must be aligned to a 4KB page. This address represents the last 4KB page of the fence region (Upper Bound is included in the fence region). Graphics Address is the offset within GMADR space.
	45:32	**Reserved** Project: All Format: MBZ
	31:12	**Fence Lower Bound** Project: All Address: GraphicsAddress[31:12] Bits 31:12 of the starting Graphics Address of the fence region. Fence regions must be aligned to 4KB. This address represents the first 4KB page of the fence region (Lowe Bound is included in the fence region). Graphics Address is the offset within GMADR space.
	11:2	**Fence Pitch** Project: All Default Value: 0h DefaultVaueDesc Format: U10-1 Width in 128 bytes This field specifies the width (pitch) of the fence region in multiple of “tile width”. For Tile X this field must be programmed to a multiple of 512B (“003” is the minimum value) and for Tile Y this field must be programmed to a multiple of 128B (“000” is the minimum value). 000h = 128B 001h = 256B ... 3FFh = 128KB
	1	**Tile Walk** Project: All Format: MI_TileWalk This field specifies the spatial ordering of QWords within tiles.

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

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

8.20 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.

§§

9 Memory Interface Commands for Rendering Engine

9.1 Introduction

This chapter describes the formats of the “Memory Interface” commands, including brief descriptions of their use. The functions performed by these commands are discussed fully in the *Memory Interface Functions* Device Programming Environment chapter.

This chapter describes MI Commands for the original graphics processing engine. The term “for Rendering Engine” in the title has been added to differentiate this chapter from a similar one describing the MI commands for the Media Decode Engine.

The commands detailed in this chapter are used across products within the Gen4 family. However, slight changes may be present in some commands (i.e., for features added or removed), or some commands may be removed entirely. Refer to the *Preface* chapter for product specific summary.

9.2 MI_ARB_CHECK

MI_ARB_CHECK			
Project:	All	**Length Bias:**	1

The MI_ARB_CHECK instruction is used to check the ring buffer double buffered head pointer (register UHPTR). This instruction can be used to pre-empt the current execution of the ring buffer. Note that the valid bit in the updated head pointer register needs to be set for the command streamer to be pre-empted.

Programming Note:

- The current head pointer is loaded with the updated head pointer register independent of the location of the updated head
- If the current head pointer and the updated head pointer register are equal, hardware will automatically reset the valid bit corresponding to the UHPTR
- This instruction can be placed only in a ring buffer, never in a batch buffer.
- For pre-emption, the wrap count in the ring buffer head register is no longer maintained by hardware. The hardware updates the wrap count to the value in the UHPTR register.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 05h MI_ARB_CHECK Format: OpCode
	22:0	**Reserved** Project: All Format: MBZ

9.3 MI_BATCH_BUFFER_END

MI_BATCH_BUFFER_END			
Project:	All	**Length Bias:**	1

The MI_BATCH_BUFFER_END command is used to terminate the execution of commands stored in a *batch buffer* initiated using a MI_BATCH_BUFFER_START command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Ah MI_ BATCH_BUFFER_END Format: OpCode
	22:0	**Reserved** Project: All Format: MBZ

9.4 MI_BATCH_BUFFER_START

MI_BATCH_BUFFER_START			
Project:	All	**Length Bias:**	2

The MI_BATCH_BUFFER_START command is used to initiate the execution of commands stored in a *batch buffer.* For restrictions on the location of batch buffers, see Batch Buffers in the Device Programming Interface chapter of *MI Functions*.

The batch buffer can be specified as secure or non-secure, determining the operations considered valid when initiated from within the buffer and any attached (chained) batch buffers. See Batch Buffer Protection in the Device Programming Interface chapter of *MI Functions*.

Programming Notes:

- Batch buffers referenced with physical addresses must not extend beyond the end of the starting physical page (can't span physical pages). However, a batch buffer initiated using a physical address can chain to another buffer in another physical page.
- A batch buffer initiated with this command must end either with a MI_BATCH_BUFFER_END command or by chaining to another batch buffer with an MI_BATCH_BUFFER_START command.
- For virtual batch buffers, it is essential that the address location beyond the current page be populated inside the GTT. HW performs over-fetch of the command addresses and any over-fetch requires a valid TLB entry. A single extra page beyond the batch buffer is sufficient.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 31h MI_BATCH_BUFFER_START Format: OpCode
	22:12	**Reserved** Project: All Format: MBZ
	11	**Reserved** Project: All Format: MBZ

MI_BATCH_BUFFER_START

10:9 Command Arbitration Control

This field controls where command arbitration can occur during the batch buffer.

Value	Name	Description	Project
0h	Arbitrate only at chain points	Legacy Mode. Overridden by MI_ARB_ON_OFF = Off	All
1h	Arbitrate between commands	Arbitration can occur between any pair of commands, or during execution of a primitive command. Overridden by MI_ARB_ON_OFF = Off	All
2h	Reserved		All
3h	No Arbitration	The Batch Buffer execution cannot be pre-empted until control returns to the initiating ring. I.e., command arbitration does not occur during or between batch buffer chains. This avoids software from having to place MI_ARB_ON_OFF packets around batch buffers to prevent interruption.	All

8 Reserved Project: All Format: MBZ

Although **Buffer Security Indicator** *is* implemented, there is no usage model for it and it need not be validated.

7 Memory Space Select

Project: All

Specifies memory space associated with the **Buffer Start Address.**

Value	Name	Description	Project
0h	Physical Memory	Physical Main (unsnooped) Memory. The 4 bits of the **Batch Buffer Start Address Extension** are prefixed to bits 31:6 of **Buffer Start Address** to specify an address within physical main memory. In this mode the hardware must not fetch data beyond a 4KB boundary.	All
1h	Graphics Memory	(GTT-mapped) Bits 31:2 of a graphics memory address. The GGTT is used to translate this address.	All

Programming Notes	Project
Batch buffers referenced with physical addresses must not extend beyond the end of the starting physical page (can't span physical pages). However, a batch buffer initiated using a physical address can chain to another buffer in another physical page.	All
Batch buffers can chain between (but cannot span) memory spaces.	All

6 Reserved Project: All Format: MBZ

MI_BATCH_BUFFER_START		
	5:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total - Bias
1	31:6	**Batch Buffer Start Address** Project: All Address: SelectableAddress(Memory Space Select)[31:6] Surface Type: BatchBuffer This field specifies Bits 31:6 of the starting address of the 64B aligned batch buffer. The address space used depends on *Memory Space Select* (see above).
	5:4	**Reserved** Project: All Format: MBZ
	3:0	**Batch Buffer Start Address Extension** Project: All Address: PhysicalAddressExtension[35:32] This field specifies bits 35:32 of the starting address of the 64B-aligned physical batch buffer. This field must be zero for non-physical Batch Buffers.

9.5 MI_DISPLAY_FLIP

MI_DISPLAY_FLIP			
Project:	All	**Length Bias:**	2

The MI_DISPLAY_FLIP command is used to request a specific display plane to switch (flip) to display a new buffer. The buffer is specified with a starting address and pitch. The tiled attribute of the buffer start address is programmed as part of the packet.

The operation this command performs is also known as a "display flip request" operation – in that the flip operation itself will occur at some point in the future. This command specifies when the flip operation is to occur: either synchronously with vertical retrace to avoid tearing artifacts (possibly on a future frame), or asynchronously (as soon as possible) to minimize rendering stalls at the cost of tearing artifacts.

Programming Notes:

1. Prior to a display flip operation being requested, software must ensure that the new display buffer is coherent in memory. This will typically require MI_DISPLAY_FLIP to be included in a PIPE_CONTROL command to flush pending rendering operations and any pending write buffers/caches, although the use of an MI_FLUSH command will also suffice albeit with greater performance penalty. (Note that completion of the MI_FLUSH command does not guarantee that previous outstanding flip operations have completed).
2. This command simply requests a display flip operation -- command execution then continues normally. There is no guarantee that the flip (even if asynchronous) will occur prior to subsequent commands being executed. (Note that completion of the MI_FLUSH command does not guarantee that outstanding flip operations have completed). The MI_WAIT_FOR_EVENT command can be used to provide this synchronization – by pausing command execution until a pending flip has actually completed. This synchronization can also be performed by use of the Display Flip Pending hardware status. See Display Flip Synchronization in the Device Programming Interface chapter of *MI Functions*.
3. After a display flip operation is requested, software is responsible for initiating any required synchronization with subsequent buffer clear or rendering operations. For multi-buffering (e.g., double buffering) operations, this will typically require updating SURFACE_STATE or the binding table to change the rendering (back) buffer. In addition, prior to any subsequent clear or rendering operations, software must typically ensure that the new rendering buffer is not actively being displayed. Again, the MI_WAIT_FOR_EVENT command or Display Flip Pending hardware status can be used to provide this synchronization. See Display Flip Synchronization in the Device Programming Interface chapter of *MI Functions.*
4. The display buffer command uses the X and Y offset for the tiled buffers from the Display Interface registers. Software is allowed to change the offset via the MMIO interface irrespective of the flip commands enqueued in the command stream. For tiled buffers, the display subsystem uses the X and Y offset in generation of the final request to memory. The offset is always updated on the next vblank for both Synchronous and Asynch Flips. It is not necessary to have a flip enqueued to update the X and Y offset
5. The display buffer command uses the linear dword offset for the linear buffers from the Display Interface registers. Software is allowed to change the offset via the MMIO interface irrespective of the flip commands enqueued in the command stream. For linear buffers, the display subsystem uses the dword offset in generation of the final request to memory.
   - For synchronous flips the offset is updated on the next vblank. It is not necessary to have a sync flip enqueued to update the dword offset.
   - Linear memory does not support asynchronous flips

MI_DISPLAY_FLIP

6. DWord 3 (panel fitter flip) must not be sent with asynchronous flips. It is only allowed to be sent with synchronous flips.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 14h MI_DISPLAY_FLIP Format: OpCode
	22	**Asynchronous Flip** Project: All Format: Boolean This field specifies whether the flip operation should be performed asynchronously to vertical retrace. If FALSE, the flip will occur during the vertical blanking interval – thus avoiding any tearing artifacts. If TRUE, the flip will occur "as soon as possible" – and may exhibit tearing artifacts **Value** **Name** **Description** **Project** 0h Asynchronous Flip All 1h Synchronous Flip All **Programming Notes** **Project** • This command must not be used to perform an Asynchronous Flip to the same address as specified in the previously executed Asynchronous Flip, or the device operation is UNDEFINED. All • The **Display Buffer Pitch and Tile parameter** fields are ignored for asynchronous flips (i.e., the new buffer must have the same pitch/tile format as the previous buffer). • **Supported on X-Tiled Frame buffers only.** • For Asynch Flips the Buffers used must be 32KB aligned. • The display stride must be >=8KB when doing Asynch Flips together with 180 display rotation. • The display stride must be power of 2 when doing Asynch Flips. • Supported on Display Planes A and B only • Not supported via the flip queue (if this bit is set, Flip Queue Select must be 0)

MI_DISPLAY_FLIP

	21:20	**Display (Plane) Select** Project: All Format: U2 This field selects which display plane is to perform the flip operation. **Value** **Name** **Description** **Project** 0h Display Plane A All 1h Display Plane A All 2h Display Plane C All 3h Display Sprite A Reserved 3h Reserved All 3h Display Sprite B Reserved
	19:6	**Reserved** Project: All Format: MBZ
	5:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1		
	31:30	**Reserved** Project: All Format: MBZ
	29	**Flip Queue Select** Project: All This field selects whether this flip is placed in the flip queue or is a standard (legacy) flip request. **Value** **Name** **Description** **Project** 0h Standard Flip Use standard (legacy) synchronous or asynchronous flipping All 1h Enqueue Flip Enqueue Flip (see *Display Functions* for a description of the Flip Queue) All **Programming Notes** **Project** Performing a legacy synchronous or asynchronous flip will drop any outstanding flips in the flip queue as well as any previous synchronous flip that has not yet completed. All
	28:15	**Reserved** Project: All Format: MBZ

MI_DISPLAY_FLIP

	14:3	**Display Buffer Pitch** Project: All Default Value: 0h DefaultVaueDesc Format: U12 Quad Words For Synchronous or Queued Flips only, this field specifies the QWord pitch of the new display buffer. For Asynchronous Flips, this parameter is ignored. All the flips in a flip chain should maintain the same pitch as programmed with the last synchronous flip or direct thru mmio.
	2:0	**Reserved** Project: All Format: MBZ
2	31:12	**Display Buffer Base Address** Project: All Address: GraphicsAddress[31:12] This field specifies Bits 31:12 of the Graphics Address of the new display buffer. The display buffer must be pixel aligned within the Graphics Address space. (Refer to the Display Address Start Address Register description in the *Display Registers* chapter). **Programming Notes** • The Display buffer must reside completely in Main Memory • This address is always translated via the *global* (rather than per-process) GTT
	11:1	**Reserved** Project: All Format: MBZ
	0	**Tile Parameter** Project: All Default Value: 0h DefaultVaueDesc Address: GraphicsAddress[31:0] For Asynchronous Flips, this parameter is ignored. All the flips in a flip chain should maintain the same tile parameter as programmed with the last synchronous flip or direct thru mmio. For Synchronous Flips, tile parameter can change for different flips in the flip chain **Value** **Name** **Description** **Project** 0h Linear For Syncronous Flips Only All 1h Tiled X All
3	31	**Enable Panel Fitter** Project: All Format: Enable Enables the panel fitter on the pipe attached to the plane selected for this flip.

MI_DISPLAY_FLIP

	30	**Panel Fitter Select** Project: All **Value** **Name** **Description** **Project** 0h 7x5 Select 7x5 capable panel fitter All 1h 3x3 Select 3x3 capable panel fitter All
	29:28	**Reserved** Project: All Format: MBZ
	27:16	**Pipe Horizontal Source Image Size** Project: All Format: U32 This 12-bit field specifies Horizontal source image size up to 4096. This determines the size of the image created by the display planes sent to the blender. The value programmed should be the source image size minus one. This field obeys all the rules of the Horizontal Source Image Size registers. The pipe affected will be the pipe attached to the plane selected for this flip.
	15:12	**Reserved** Project: All Format: MBZ
	11:0	**Pipe Vertical Source Image ReSize** Project: All Format: U32 This 12-bit field specifies the new vertical source image size up to 4096 lines. This determines the size of the image created by the display planes sent to the blender. The value programmed should be the source image size minus one. This field obeys all the rules of the Vertical Source Image Size registers. The pipe affected will be the pipe attached to the plane selected for this flip.

9.6 MI_FLUSH

MI_FLUSH

Project:	All	Length Bias:	1

The MI_FLUSH command is used to perform an internal "flush" operation. The parser pauses on an internal flush until all drawing engines have completed any pending operations and the read caches are invalidated including the texture cache accessed via the Sampler or the data port. In addition, this command can also be used to:

1. Flush any dirty data in the Render Cache to memory. This is done by default, however this can be inhibited.
2. Invalidate the state and command cache.

Usage note: After this command is completed and followed by a Store DWord-type command, CPU access to graphics memory will be coherent (assuming the Render Cache flush is not inhibited).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 04h MI_FLUSH Format: OpCode
	22:6	**Reserved** Project: All Format: MBZ
	5:4	**Reserved** Project: All Format: MBZ
	3	**Global Snapshot Count Reset** Project: All Format: Boolean If set, the snapshot registers defined for the Gen4 debug capability are reset after the flush completes. The Statistics Counters are also reset; SW should never set this bit during normal operation since the Statistics Counters are intended to be free running. **Programming Notes** — **Project** PS_DEPTH_COUNT and TIMESTAMP are *not* reset by MI_FLUSH with this bit set. TIMESTAMP and PS_DEPTH_COUNT can be reset by writing 0 to them — All **Value** / **Name** / **Description** / **Project** 0h / Don't Reset / Do not reset the snapshot counts or Statistics Counters. / All 1h / Reset / Reset the snapshot count in Gen4 for all the units and reset the Statistics Counters except as noted above. / All
	2	**Render Cache Flush Inhibit** Project: All Format: Boolean If set, the Render Cache is not flushed as part of the processing of this command. **Value** / **Name** / **Description** / **Project** 0h / Flush / Flush the Render Cache / All 1h / Don't Flush / Do not flush the Render Cache / All

MI_FLUSH		
	1	**State/Instruction Cache Invalidate** Project: All Format: Boolean If set, Invalidates the State and Instruction Cache **Value** / **Name** / **Description** / **Project** 0h / Don't Invalidate / Leave State/Instruction Cache unaffected / All 1h / Invalidate / Invalidate State/Instruction Cache / All
	0	**Reserved** Project: All Format: MBZ

9.7 MI_LOAD_REGISTER_IMM

MI_LOAD_REGISTER_IMM			
Project:	All	**Length Bias:**	2

The MI_LOAD_REGISTER_IMM command requests a write of up to a DWord constant supplied in the command to the specified Register Offset (i.e., offset into Memory-Mapped Register Range). The register is loaded before the next command is executed.

Programming Notes:

The behavior of this command is controlled by Dword 3, Bit 8 (**Disable Register Access**) of the RINGBUF register. If this command is disallowed then the command stream converts it to a NOOP.

If this command is executed from a BB then the behavior of this command is controlled by Dword 0, Bit 8 (Security Indicator) of the BATCH_BUFFER_START Command. If the batch buffer is insecure then the command stream converts this command to a NOOP. Note that the corresponding ring buffer must allow a register update for this command to execute.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 22h MI_ Format: OpCode
	22:12	**Reserved** Project: All Format: MBZ
	11:8	**Byte Write Disables** Format: Enable[4] Bit 8 corresponds to Data DWord [7:0] Range Must specify a valid register write operation This field specifies which bytes of the **Data DWord** are **not** to be written to the DWord offset specified in *Register Offset.*
	7:6	**Reserved** Project: All Format: MBZ

MI_LOAD_REGISTER_IMM		
	5:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:2	**Register Offset** Format: U30 Address: MmioAddress[31:2] This field specifies bits [31:2] of the offset into the Memory Mapped Register Range (i.e., this field specifies a DWord offset).
	1:0	**Reserved** Project: All Format: MBZ
2	31:0	**Data DWord** Mask: Bytes Write Disables Format: U32 This field specifies the DWord value to be written to the targeted location.

9.8 MI_LOAD_SCAN_LINES_EXCL

MI_LOAD_SCAN_LINES_EXCL			
Project:	All	**Length Bias:**	2

The MI_LOAD_SCAN_LINES_EXCL command is used to initialize the Scan Line Window registers for a specific Display Pipe. If the display refresh is ***outside*** this window the Display Engine asserts a signal that is used by the command parser to process the WAIT_FOR_EVENT command (i.e., the parser will wait while outside). This command overrides the Scan Line Window defined by any previous MI_LOAD_SCAN_LINES_INCL or MI_LOAD_SCAN_LINES_EXCL commands targeting the specific display pipe.

Note: The two scan-line numbers are inclusive. If programmed to the same values, that single line defines the region in question.

Always place an even number of MI_LOAD_SCAN_LINES_EXCL/INCL at a time into the ring buffer. If only a single MI_LOAD_SCAN_LINES_EXCL/INCL is desired, just add a second identical MI_LOAD_SCAN_LINES_EXCL/INCL command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 13h MI_LOAD_SCAN_LINES_EXCL Format: OpCode
	22	**Reserved** Project: All Format: MBZ

MI_LOAD_SCAN_LINES_EXCL

	21:20	**Display Pipe Select** Project: All Format: U2 This field selects which Display Engine (pipe) this command is targeting. **Value** **Name** **Description** **Project** 0h Display Pipe A All 1h Display Pipe B All
	19:6	**Reserved** Project: All Format: MBZ
	5:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:16	**Start Scan Line Number** Project: All Format: U16 In scan lines, where scan line 0 is the first line of the display frame. Range [0,Display Buffer height in lines-1] This field specifies the starting scan line number of the Scan Line Window.
	31:16	**End Scan Line Number** Project: All Format: U16 In scan lines, where scan line 0 is the first line of the display frame. Range [0,Display Buffer height in lines-1] This field specifies the ending scan line number of the Scan Line Window.

9.9 MI_LOAD_SCAN_LINES_INCL

MI_LOAD_SCAN_LINES_INCL			
Project:	All	**Length Bias:**	2

The MI_LOAD_SCAN_LINES_INCL command is used to initialize the Scan Line Window registers for a specific Display Engine. If the display refresh is ***within*** this window the Display Engine asserts a signal that is used by the command parser to process the WAIT_FOR_EVENT command (i.e., the parser will wait while inside of the window). This command overrides the Scan Line Window defined by any previous MI_LOAD_SCAN_LINES_INCL or MI_LOAD_SCAN_LINES_EXCL commands targeting the specific display.

Always place an even number of MI_LOAD_SCAN_LINES_EXCL/INCL at a time into the ring buffer. If only a single MI_LOAD_SCAN_LINES_EXCL/INCL is desired, just add a second identical

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 12h MI_LOAD_SCAN_LINES_INCL Format: OpCode
	22	**Reserved** Project: All Format: MBZ
	21:20	**Display Pipe Select** Project: All Format: U2 This field selects which Display Engine (pipe) this command is targeting. **Value** **Name** **Description** **Project** 0h Display Pipe A All 1h Display Pipe B All
	19:6	**Reserved** Project: All Format: MBZ
	5:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:16	**Start Scan Line Number** Project: All Format: U16 In scan lines, where scan line 0 is the first line of the display frame. Range [0,Display Buffer height in lines-1] This field specifies the starting scan line number of the Scan Line Window.

MI_LOAD_SCAN_LINES_INCL		
	31:16	**End Scan Line Number** Project: All Format: U16 In scan lines, where scan line 0 is the first line of the display frame. Range [0,Display Buffer height in lines-1] This field specifies the ending scan line number of the Scan Line Window.

9.10 MI_NOOP

MI_NOOP			
Project:	All	**Length Bias:**	1

The MI_NOOP command basically performs a “no operation” in the command stream and is typically used to pad the command stream (e.g., in order to pad out a batch buffer to a QWord boundary). However, there is one minor (optional) function this command can perform – a 22-bit value can be loaded into the MI NOPID register. This provides a general-purpose command stream tagging ("breadcrumb") mechanism (e.g., to provide sequencing information for a subsequent breakpoint interrupt).

Performance Note: The process time to execute a NOP command is min of 6 clock cycles. One example usage of the improved NOP throughput is for some multi-pass media application whereas some unwanted media object commands are replaced by MI_NOOP without repacking the commands in a batch buffer.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0h MI_NOOP Format: OpCode
	22	**Identification Number Register Write Enable** Project: All Format: Enable This field enables the value in the Identification Number field to be written into the MI NOPID register. If disabled, that register is unmodified – making this command an effective “no operation” function. **Value** **Name** **Description** **Project** 0h Disable Do not write the NOP_ID register. All 1h Enable Write the NOP_ID register. All
	31:0	**Identification Number** Project: All Format: U22 This field contains a 22-bit number which can be written to the MI NOPID register.

9.11 MI_OVERLAY_FLIP

MI_OVERLAY_FLIP			
Project:	All	Length Bias:	2

The MI_OVERLAY_FLIP command is used to specify memory buffers that will (optionally) be used during the next Vertical Blank period to update the specified Overlay control register set and Overlay filter coefficients (respectively). The update of the Overlay registers is referred to as an "Overlay Flip", making this command an "Overlay Flip Request".

Programming Notes:

1. Prior to an overlay flip operation being requested, software must ensure that the memory buffer used to update the overlay registers is coherent (i.e., there are no outstanding buffered writes to that memory buffer).
2. Prior to an overlay flip operation being requested, software must ensure that the new overlay buffer is coherent in memory. This will typically require the use of an MI_FLUSH command to flush pending rendering operations and any pending write buffers/caches.
3. This command simply requests an overlay flip operation -- command execution then continues normally. There is no mechanism to prevent a new flip request from overriding any outstanding flip request. (Note that completion of the MI_FLUSH command does not guarantee that outstanding flip operations have completed). The MI_WAIT_FOR_EVENT command can be used to provide this synchronization – by pausing command execution until a pending overlay flip has actually completed or that the display refresh has proceeded past a specific scan line window. This synchronization can also be performed by use of the Overlay Flip Pending hardware status. See Overlay Flip Synchronization in the Device Programming Interface chapter of *MI Functions*.
4. After an overlay flip operation is requested, software is responsible for initiating any required synchronization with subsequent buffer clear or rendering operations targeting the previous ("flipped-from") overlay buffer.
5. Registers and Coefficients are located in Main memory.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 11h MI_OVERLAY_FLIP Format: OpCode

MI_OVERLAY_FLIP

	22:21	**Mode Flags** Project: All Format: U2 **Value** / **Name** / **Description** / **Project** 00b / Flip Continue / Do not flush or change the state of the Render Cache or Overlay. / All 01b / Flip On / Flush Render Cache, drawing pipeline and then set render cache in overlay Mode before executing the Flip. The Flip turns on the overlay engine. This Render Cache flush is not applicable in a Mobile Gfx controller which has an independent overlay data buffer. / All 10b / Flip Off / Flush Render Cache, drawing pipeline and then clear Overlay Mode and turn off the overlay engine. Do not update registers and coefficients from memory. This Render Cache flush is required because overlay shares the render cache in desktop graphics controllers. This bit is generally not applicable in a Mobile graphics controller which has an independent overlay data buffer. / All 11b / Reserved / All
	20:6	**Reserved** Project: All Format: MBZ
	5:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:12	**Register and Coefficient Update Address** Project: All Address: GlobalGraphicsAddress[31:12] Surface Type: U32 This field specifies the memory buffer used to update the overlay registers and Coefficients. The Overlay Update Address Register specifies a **Global GTT** address used by the Overlay at the next VBLANK event to start requesting overlay control register and Coefficient data from memory. Software should ensure that the **Global GTT** address is **page-aligned**, so that the entire overlay control registers and coefficients are within one 4K page.
	11:1	**Reserved** Project: All Format: MBZ

MI_OVERLAY_FLIP		
	0	**Overlay Filter Coefficient Register Update Flag (OFC_UPDATE)** Project: All This field indicates if hardware should load overlay filter coefficients from memory. Turning overlay off without loading the Overlay Filter Coefficient registers via MI_OVERLAY_FLIP can lead to a hang. **Value** / **Name** / **Description** / **Project** 0h / Don't Update / Do not update overlay filter coefficients. / All 1h / Update / Hardware loads the overlay filter coefficients from memory to on-chip registers. / All

9.11.1 Turning the Overlay Off

The Overlay Engine is turned off by issuing an MI_OVERLAY_FLIP with the **Mode Flags** set to '10'b (aka "Flip Off), thereby flushing and reconfiguring the internal caches and putting the Overlay Engine into a low-power state. Software must ensure that the subsequent Overlay Flip has occurred at the next associated VBlank, typically by use of the **Overlay Flip Pending Wait Enable** bit of the MI_WAIT_FOR_EVENT command. In addition, the Display Pipe to which the overlay is attached must continue running until the sequence completes, or device operation is UNDEFINED.

In order to completely shutdown the Overlay Engine, and additional step is required before the use of the "Flip Off" sequence (as described above). The Overlay Enable (OV_ENBL) bit of the Overlay Command (OCOMD) Register must be cleared via a normal Overlay Register load accomplished via issuance of an MI_OVERLAY_FLIP with Mode Flags = '00'b (aka Flip Continue). This operation will effectively turn off the display of the overlay. Note that a wait-for-overlay-VBlank must be used to ensure this Flip Continue has completed. The subsequent Flip-Off sequence (above) will reconfigure the cache for non-overlay operation and gracefully power down the Overlay Engine.

9.11.2 Valid Overlay Flip Sequences

The only architecturally valid Overlay Flip sequence is shown below:

- FlipOn
- some number of FlipContinues
- FlipOff

For example, multiple FlipOn commands (without intervening FlipOff commands) are invalid; multiple FlipOff commands (without intervening FlipOn commands) are invalid; FlipContinue without a preceding FlipOn is invalid.

9.12 MI_REPORT_HEAD

MI_REPORT_HEAD			
Project:	All	**Length Bias:**	1

The MI_REPORT_HEAD command causes the Head Pointer value of the active ring buffer to be written to a cacheable (snooped) system memory location.

The location written is relative to the address programmed in the Hardware Status Page Address Register.

Programming Notes:

This command must not be executed from a Batch Buffer (Refer to the description of the HSW_PGA register).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 07h MI_REPORT_HEAD Format: OpCode
	22:0	**Reserved** Project: All Format: MBZ

9.13 MI_SET_CONTEXT

MI_SET_CONTEXT			
Project:	All	**Length Bias:**	2

The MI_SET_CONTEXT command is used to specify the *logical* context associated with the hardware context. A logical context is an area in memory used to store hardware context information, and the context is referenced via a 2KB-aligned pointer. If the (new) logical context is different (i.e., at a different memory address), the device will proceed to save the current HW context values to the current logical context address, and then restore (load) the new logical context by reading the context from the new address and loading it into the hardware context state. If the logical context address specified in this command matches the current logical context address, this command is effectively treated as a NOP.

This command also includes some controls over the context save/restore process.

- The **Force Restore** bit can be used to refresh the on-chip device state from the same memory address if the indirect state buffers have been modified.
- The **Restore Inhibit** bit can be used to prevent the new context from being loaded at all. This **must** be used to prevent an uninitialized context from being loaded. Once software has initialized a context (by setting all state variables to initial values via commands), the context can then be stored and restored normally.
- This command needs to be always followed by a single MI_NOOP instruction to workaround a Gen4 silicon issue.

MI_SET_CONTEXT

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 18h MI_SET_CONTEXT Format: OpCode
	22:6	**Reserved** Project: All Format: MBZ
	5:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:11	**Logical Context Address** Project: All Address: PhysicalAddress[31:11] Surface Type: Logical Context This field contains the 2KB-aligned physical address of the Logical Context that is to be loaded into the hardware context. If this address is equal to the CCID register associated with the current ring, no load will occur. Prior to loading this new context, the device will save the existing context as required. After the context switch operation completes, this address will be loaded into the associated CCID register.
	10	**Reserved** Project: All Format: MBZ
	9	**HD DVD Context** Project: All **Value** **Name** **Description** **Project** 0h Regular Context All 1h HD DVD Context Special considerations for TDP allow for higher voltage and frequency. All

MI_SET_CONTEXT

	8	**Memory Space Select** Project: All BitFieldDesc **Value** / **Name** / **Description** / **Project** 0h / Physical Memory / Physical Main (unsnooped) Memory. The 4 bits of **Physical Start Address Extension** are prefixed to bits 31:11 to specify a 2KB aligned address within physical main memory. In this mode the hardware must not fetch data beyond a 4KB boundary. / All 1h / Global Graphics Memory / Global Graphics (GTT-mapped) Memory. Bits 31:11 of a graphics memory address. The GTT whose address is contained in the PGTBL_CTL register is used to translate this address. / All
	7:4	**Logical Context Address Extension** Project: All Address: PhysicalAddressExtension[35:32] Surface Type: Logical Context This field specified Bits 35:32 of the starting address of the 2KB-aligned physical logical context address. This field must be zero for global gtt context address.
	3	**Extended State Save Enable** Project: All Format: U32 If set, the extended state identified in the Logical Context Data section of the Memory Data Formats chapter is saved as part of switching away from this logical context. This bit will be stored in the associated CCID register to control the context save operation when switching away from this context (as part of a subsequent MI_SET_CONTEXT command).
	2	**Extended State Restore Enable** Project: All Format: U32 If set, the extended state identified in the Logical Context Data section of the Memory Data Formats chapter is loaded (or restored) as part of switching to this logical context. This method can be used to restore things such as filter coefficients using the indirect state restore followed by a restore of the extended logical context data. This bit affects the switch (if required) to the context specified in **Logical Context Address**. This bit will also be stored in the associated CCID register to control a subsequent context save operation when switching to this context (as part of a subsequent ring buffer switch).
	1	**Force Restore** Project: All Format: U32 When switching to this logical context a comparison between Logical Context Address and the contests of the CCID register is performed. Normally, matching addresses prevent a context restore from occurring; however, when this bit is set a context restore is forced to occur. This bit cannot be set with Restore Inhibit. **Note:** This bit is not saved in the associated CCID register. It only affects the processing of this command.

MI_SET_CONTEXT		
	0	**Restore Inhibit** Project: All Format: U32 If set, the restore of the HW context from the logical context specified by **Logical Context Address** is inhibited (i.e., the existing HW context values are maintained). This bit must be used to prevent the loading of an uninitialized logical context. If clear, the context switch proceeds normally. This bit cannot be set with Force Restore. **Note:** This bit is not saved in the associated CCID register. It only affects the processing of this command.

9.14 MI_STORE_DATA_IMM

MI_STORE_DATA_IMM			
Project:	All	**Length Bias:**	2

The MI_STORE_DATA_IMM command requests a write of the QWord constant supplied in the packet to the specified Memory Address. As the write targets a System Memory Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes:

- This command should not be used within a “non-secure” batch buffer to access per-process virtual space. Doing so will cause the command parser to perform the write with byte enables turned off. This command can be used within ring buffers and/or “secure” batch buffers.
- This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll un-cached memory or device registers).
- This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete “eventually”, there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 20h MI_STORE_DATA_IMM Format: OpCode

MI_STORE_DATA_IMM

22 — Memory Address Type

Project: All

Value	Name	Description	Project
0h	Physical Address		All
1h	Graphics Address	Hardware will translate this address using the operating GTT. The GTT (global or per-process) used for the translation will be the same GTT used to access the buffer executing this command.	All

21 — BitFieldName

Project: All

This bit will be ignored and treated as if clear when executing from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer.

Value	Name	Description	Project
0h	Per Process Graphics Address		All
1h	Global Graphics Address	This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.	All

Programming Notes

Notes

20:6 — Reserved Project: All Format: MBZ

5:0 — DWord Length

Default Value: 2h Excludes DWord (0,1) = 2 for DWord, 3 for QWord

Format: =n Total Length - 2

1 — 31:4 Reserved Project: All Format: MBZ

3:0 — Physical Start Address Extension

Project: All

Address: PhysicalAddressExtension[35:32]

Surface Type: U64

This field specifies bits 35:32 of the physical address where the data will be stored. This field must be zero for a virtual address.

MI_STORE_DATA_IMM		
2	31:2	**Address** Project: All Address: SelectableAddress(Memory Address Type) [31:2] Surface Type: U32(2) This field specifies Bits 31:2 of the Address where the DWord will be stored. As the store address must be DWord-aligned, Bits 1:0 of that address MBZ. This address must be 8B aligned for a store "QW" command. Format = U30, Range = valid System Memory Address (not mapped by GTT) if Physical Format = Bits[31:2] of a Graphics Memory Address If Virtual
	1:0	**Reserved** Project: All Format: MBZ
3	31:0	**Data DWord 0** Project: All Format: U32 This field specifies the DWord value to be written to the targeted location. For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).
4	31:0	**Data DWord 1** Project: All Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

9.15 MI_STORE_DATA_INDEX

MI_STORE_DATA_INDEX			
Project:	All	**Length Bias:**	2

- The MI_STORE_DATA_INDEX command requests a write of the data constant supplied in the packet to the specified offset from the System Address defined by the Hardware Status Page Address Register. As the write targets a System Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).
- Programming Notes:
- Use of this command with an invalid or uninitialized value in the Hardware Status Page Address Register is UNDEFINED.
- This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll uncached memory or device registers).
- This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete "eventually", there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

MI_STORE_DATA_INDEX

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 21h MI_STORE_DATA_INDEX Format: OpCode
	22:21	**Reserved** Project: All Format: MBZ
	20:6	**Reserved** Project: All Format: MBZ
	5:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) = 1 for DWord, 2 for QWord Format: =n Total Length - 2
1	31:12	**Reserved** Project: All Format: MBZ
	11:2	**Offset** Project: All Format: U10 zero-based DWord offset into the HW status page. Address: HardwareStatusPageOffset[11:2] Surface Type: U32 Range [16, 1023] This field specifies the offset (into the hardware status page) to which the data will be written. Note that the first few DWords of this status page are reserved for special-purpose data storage – targeting these reserved locations via this command is UNDEFINED.
	1:0	**Reserved** Project: All Format: MBZ
2	31:0	**Data DWord 0** Project: All Format: U32 This field specifies the DWord value to be written to the targeted location. For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).
3	31:0	**Data DWord 1** Project: All Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

9.16 MI_STORE_REGISTER_MEM

MI_STORE_REGISTER_MEM

Project:	All	Length Bias:	2

The MI_STORE_REGISTER_MEM command requests a register read from a specified memory mapped register location in the device and store of that DWord to memory. The register address is specified along with the command to perform the read.

Programming Notes:

- The command temporarily halts command execution.
- The memory address for the write is snooped on the host bus.
- This command should not be used within a “non-secure” batch buffer to access per-process virtual space. Doing so will cause the command parser to perform the write with byte enables turned off. This command can be used within ring buffers and/or “secure” batch buffers.
- This command will cause undefined data to be written to memory if given register addresses for the PGTBL_CTL_0 or FENCE registers

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 24h MI_STORE_REGISTER_MEM Format: OpCode
	22	**Reserved** Project: DevBW-A,B Format: MBZ
	22	**Memory Address Type** Project: All, except DevBW-A,B **Value** / **Name** / **Description** / **Project** 0h / Physical Address / / All 1h / Graphics Address / Hardware will translate this address using the operating GTT. The GTT (global or per-process) used for the translation will be the same GTT used to access the buffer executing this command. / All
	21:6	**Reserved** Project: All Format: MBZ
	5:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) Format: =n Total Length - 2

MI_STORE_REGISTER_MEM

1	31:28	**Physical Start Address Extension** Project: All Address: PhysicalAddressExtension[35:32] Surface Type: MMIO Register This field specifies bits 35:32 of the starting address of the physical address.
	27:19	**Reserved** Project: All Format: MBZ
	18:1	**Register Address** Project: All Address: MMIO Address[18:2] Surface Type: MMIO Register This field specifies Bits 18:2 of the Register offset the DWord will be read from. As the register address must be DWord-aligned, Bits 1:0 of that address MBZ. **Programming Notes** **Project** Storing a VGA register is not permitted and will store an UNDEFINED value. All The values of PGTBL_CTL0 or any of the FENCE registers cannot be stored to memory; UNDEFINED values will be written to memory if the addresses of these registers are specified. All
	1	**Reserved** Project: All Format: MBZ
	0	**Reserved** Project: All Format: MBZ
2	31:2	**Memory Address** Project: All Address: SelectableAddress(Memory Address Type)[31:2] Surface Type: MMIO Register This field specifies the address of the memory location where the register value specified in the DWord above will be written. The address specifies the DWord location of the data. If Memory Address Type = 0, Range = Physical_Address [31:2] If Memory Address Type = 1, Range = GraphicsMemoryAddress[31:2]
	1:0	**Reserved** Project: All Format: MBZ

9.17 MI_USER_INTERRUPT

MI_USER_INTERRUPT			
Project:	All	**Length Bias:**	1

The MI_USER_INTERRUPT command is used to generate a User Interrupt condition. The parser will continue parsing after processing this command. See User Interrupt.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 02h MI_USER_INTERRUPT Format: OpCode
	22:0	**Reserved** Project: All Format: MBZ

9.18 MI_WAIT_FOR_EVENT

MI_WAIT_FOR_EVENT			
Project:	All	**Length Bias:**	1

The MI_WAIT_FOR_EVENT command is used to pause command stream processing until a specific event occurs or while a specific condition exists. See Wait Events/Conditions, Device Programming Interface in *MI Functions*. Only one event/condition can be specified -- specifying multiple events is UNDEFINED.

The effect of the wait operation depends on the source of the command. If executed from a batch buffer, the parser will halt (and suspend command arbitration) until the event/condition occurs. If executed from a ring buffer, further processing of that ring will be suspended, although command arbitration (from other rings) will continue. Note that if a specified condition does not exist (the condition code is inactive) at the time the parser executes this command, the parser proceeds, treating this command as a no-operation.

If execution of this command from a primary ring buffer causes a wait to occur, the active ring buffer will *effectively* give up the remainder of its time slice (required in order to enable arbitration from other primary ring buffers).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 03h MI_WAIT_FOR_EVENT Format: OpCode
	22:19	**Reserved** Project: All Format: MBZ

MI_WAIT_FOR_EVENT

18	**Display Pipe B Start of V Blank Wait Enable**	Project: All Format: Enable

This field enables a wait until the start of next Display Pipe B “Vertical Blank” event occurs. This event is defined as the start of the next Display B Vertical blank period. Note that this can cause a wait for up to a frame. See Start of Vertical Blank Event in the Device Programming Interface chapter of *MI Functions*.

Errata	Description	Project
BWT013	MBZ	DevBW

17	**Display Pipe A Start of V Blank Wait Enable**	Project: All Format: Enable

This field enables a wait until the start of next Display Pipe A “Vertical Blank” event occurs. This event is defined as the start of the next Display A Vertical blank period. Note that this can cause a wait for up to a frame. See Start of Vertical Blank Event in the Device Programming Interface chapter of *MI Functions*.

Programming Notes	Project
Notes	All

Errata	Description	Project
BWT013	MBZ	DevBW

16	**Overlay Flip Pending Wait Enable**	Project: All Format: Enable

This field enables a wait for the duration of an Overlay “Flip Pending” condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new overlay address has been loaded into the corresponding overlay registers). See Overlay Flip Pending Condition in the Device Programming Interface chapter of *MI Functions*.

15	**Reserved**	Project: All Format: MBZ
14	**Display Pipe B H Blank Wait Enable**	Project: All Format: Enable

This field enables a wait until the start of next Display Pipe B “Horizontal Blank” event occurs. This event is defined as the start of the next Display B Horizontal blank period. Note that this can cause a wait for up to a line. See Horizontal Blank Event in the Device Programming Interface chapter of *MI Functions*.

13	**Display Pipe A H Blank Wait Enable**	Project: All Format: Enable

This field enables a wait until the start of next Display Pipe A “Horizontal Blank” event occurs. This event is defined as the start of the next Display A Horizontal blank period. Note that this can cause a wait for up to a line. See Horizontal Blank Event in the Device Programming Interface chapter of *MI Functions*.

12:9 **Condition Code Wait Select**

Project: All

This field enables a wait for the duration that the corresponding condition code is active. These enable select one of 15 condition codes in the EXCC register, that cause the parser to wait until that condition-code in the EXCC is cleared.

Value	Name	Description	Project
0h	Not Enabled	Condition Code Wait not enabled	All
1h-5h	Enabled	Condition Code select enabled; selects one of 5 codes, 0 – 4	All
6h-15h	Reserved		All

Programming Notes **Project**

Note that not all condition codes are implemented. The parser operation is UNDEFINED if an unimplemented condition code is selected by this field. The description of the EXCC register (*Memory Interface Registers*) lists the codes that are implemented. All

8 **Display Plane C Flip Pending Wait Enable** Project: All Format: Enable

This field enables a wait for the duration of a Display Plane C “Flip Pending” condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of *MI Functions*.

7 **Display Pipe B Vertical Blank Wait Enable** Project: All Format: Enable

This field enables a wait until the next Display Pipe B “Vertical Blank” event occurs. This event is defined as the start of the next Display Pipe B vertical blank period. Note that this can cause a wait for up to an entire refresh period. See Vertical Blank Event (See *Programming Interface*).

Programming Notes **Project**

Prior to using the MI_WAIT_FOR_EVENT command to wait on Display Pipe A/B VBlank events, the corresponding Vertical Blank Interrupt Enable (bit 17) of the corresponding PIPEASTAT (70024h) or PIPEBSTAT (71024h) register must be set. Note that this does not require an actual VBlank interrupt to be enabled. All

6 **Display Plane B Flip Pending Wait Enable** Project: All Format: Enable

This field enables a wait for the duration of a Display Plane B “Flip Pending” condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of *MI Functions*.

MI_WAIT_FOR_EVENT

5	**Display Pipe B Scan Line Window Wait Enable** Project: All Format: Enable This field enables a wait while a Display B "In Scan Line Window" condition exists. This condition is defined as the period of time the Display B refresh is inside the scan line window as specified by a previous MI_LOAD_SCAN_LINES_INCL or MI_LOAD_SCAN_LINES_EXCL command. If the Display B refresh is outside this window, or a window has not been specified, the parser proceeds, treating this command as a no-op. If the Display B refresh is currently inside this window, the parser will wait until the refresh exits the window. See Scan Line Window Condition in the Device Programming Interface chapter of *MI Functions*.
4	**Frame Buffer Compression Idle Wait Enable** Project: All Format: Enable This field enables a wait while the Frame Buffer compressor is busy. The ring that this command got executed from is removed from arbitration for the wait period and is inserted into arbitration as soon as the frame buffer compressor is idle.
3	**Display Pipe A Vertical Blank Wait Enable** Project: All Format: Enable This field enables a wait until the next Display Pipe A "Vertical Blank" event occurs. This event is defined as the start of the next Display A vertical blank period. Note that this can cause a wait for up to an entire refresh period. See Vertical Blank Event in the Device Programming Interface chapter of *MI Functions*. **Programming Notes** **Project** Prior to using the MI_WAIT_FOR_EVENT command to wait on Display Pipe A/B VBlank events, the corresponding Vertical Blank Interrupt Enable (bit 17) of the corresponding PIPEASTAT (70024h) or PIPEBSTAT (71024h) register must be set. Note that this does not require an actual VBlank interrupt to be enabled. All
2	**Display Plane A Flip Pending Wait Enable** Project: All Format: Enable This field enables a wait for the duration of a Display Plane A "Flip Pending" condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of *MI Functions*.
1	**Display Pipe A Scan Line Window Wait Enable** Project: All Format: Enable This field enables a wait while a Display Pipe A "In Scan Line Window" condition exists. This condition is defined as the period of time the Display A refresh is inside the scan line window as specified by a previous MI_INCLUSIVE_SCAN_WINDOW or MI_EXCLUSIVE_SCAN_WINDOW command. If the Display A refresh is outside this window, or a window has not been specified, the parser proceeds, treating this command as a no-op. If the Display A refresh is currently inside this window, the parser will wait until the refresh exits the window. See Scan Line Window Condition in the Device Programming Interface chapter of *MI Functions*.
0	**Reserved** Project: All Format: MBZ

§§

10 Memory Interface Commands for Blitter Engine

10.1 Introduction

This chapter describes the formats of the “Memory Interface” commands, including brief descriptions of their use. The functions performed by these commands are discussed fully in the *Memory Interface Functions* Device Programming Environment chapter.

This chapter describes MI Commands for the blitter graphics processing engine. The term “for Blitter Engine” in the title has been added to differentiate this chapter from a similar one describing the MI commands for the Media Decode Engine and the Rendering Engine.

The commands detailed in this chapter are used across products within the Gen4 family. However, slight changes may be present in some commands (i.e., for features added or removed), or some commands may be removed entirely. Refer to the *Preface* chapter for product specific summary.

10.2 MI_LOAD_REGISTER_IMM

MI_LOAD_REGISTER_IMM			
Project:	All	**Length Bias:**	2

The MI_LOAD_REGISTER_IMM command requests a write of up to a DWord constant supplied in the command to the specified Register Offset (i.e., offset into Memory-Mapped Register Range). The register is loaded before the next command is executed.

Programming Notes:

The behavior of this command is controlled by Dword 3, Bit 8 (**Disable Register Access**) of the RINGBUF register. If this command is disallowed then the command stream converts it to a NOOP.

If this command is executed from a BB then the behavior of this command is controlled by Dword 0, Bit 8 (Security Indicator**) of the BATCH_BUFFER_START Command. If the batch buffer is insecure then the command stream converts this command to a NOOP. Note that the corresponding ring buffer must allow a register update for this command to execute.**

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: h MI_ Format: OpCode
	22:12	**Reserved** Project: All Format: MBZ
	11:8	**Byte Write Disables** Format: Enable[4] Bit 8 corresponds to Data DWord [7:0] Range Must specify a valid register write operation This field specifies which bytes of the **Data DWord** are **not** to be written to the DWord offset specified in *Register Offset.*
	7:6	**Reserved** Project: All Format: MBZ
	5:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:2	**Register Offset** Format: U30 Address: MmioAddress[31:2] This field specifies bits [31:2] of the offset into the Memory Mapped Register Range (i.e., this field specifies a DWord offset).
	1:0	**Reserved** Project: All Format: MBZ
2	31:0	**Data DWord** Mask: Bytes Write Disables Format: U32 This field specifies the DWord value to be written to the targeted location.

10.3 MI_NOOP

MI_NOOP			
Project:	All	**Length Bias:**	1

The MI_NOOP command basically performs a “no operation” in the command stream and is typically used to pad the command stream (e.g., in order to pad out a batch buffer to a QWord boundary). However, there is one minor (optional) function this command can perform – a 22-bit value can be loaded into the MI NOPID register. This provides a general-purpose command stream tagging ("breadcrumb") mechanism (e.g., to provide sequencing information for a subsequent breakpoint interrupt).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0h MI_NOOP Format: OpCode
	22	**Identification Number Register Write Enable** Project: All Format: Enable This field enables the value in the Identification Number field to be written into the MI NOPID register. If disabled, that register is unmodified – making this command an effective “no operation” function. **Value** **Name** **Description** **Project** 0h Disable Do not write the NOP_ID register. All 1h Enable Write the NOP_ID register. All
	31:0	**Identification Number** Project: All Format: U22 This field contains a 22-bit number which can be written to the MI NOPID register.

10.4 MI_STORE_DATA_IMM

MI_STORE_DATA_IMM			
Project:	All	**Length Bias:**	2

The MI_STORE_DATA_IMM command requests a write of the QWord constant supplied in the packet to the specified Memory Address. As the write targets a System Memory Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes:

- This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll un-cached memory or device registers). However, the cacheable nature of the transaction is determined by the setting of the "mapping type" in the GTT entry.
- This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete "eventually", there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations. All writes to memory generated using this command are expected to finish in order.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 20h MI_STORE_DATA_IMM Format: OpCode
	22	**Memory Address Type** Project: All **Value** / **Name** / **Description** / **Project** 0h / Reserved / Physical address / All 1h / Reserved / Virtual address. Hardware will translate this address using the GTT. The GTT (global or per-process) used for the translation will be the same GTT used to access the buffer executing this instruction translate this address using the GTT. The GTT (global or per-process) used for the translation will be the same GTT used to access the buffer executing this instruction. / All
	21:6	**Reserved** Project: All Format: MBZ
	5:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) = 2 for DWord, 3 for QWord Format: =n Total Length - 2
1	31:0	**Reserved** Project: All Format: MBZ
2	31:0	**Reserved** Project: All Format: MBZ

MI_STORE_DATA_IMM		
3	31:0	**Data DWord 0** Project: All Format: U32 This field specifies the DWord value to be written to the targeted location. For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).
4	31:0	**Data DWord 1** Project: All Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

10.5 MI_STORE_DATA_INDEX

MI_STORE_DATA_INDEX			
Project:	All	**Length Bias:**	2

The MI_STORE_DATA_INDEX command requests a write of the data constant supplied in the packet to the specified offset from the System Address defined by the Hardware Status Page Address Register. As the write targets a System Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes:

- Use of this command with an invalid or uninitialized value in the Hardware Status Page Address Register is UNDEFINED.
- This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll uncached memory or device registers).
- This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete “eventually”, there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 21h MI_STORE_DATA_INDEX Format: OpCode
	22	**Reserved** Project: All Format: Setting this bit will cause this command to offset in the Surface Probe List instead of the hardware status page. This is intended to be used internally only (it is UNDEFINED to set this bit in a command in a ring or batch buffer.)
	21:6	**Reserved** Project: All Format: MBZ
	5:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) = 1 for DWord, 2 for QWord Format: =n Total Length - 2

MI_STORE_DATA_INDEX		
1	31:12	**Reserved** Project: All Format: MBZ
	11:2	**Offset** Project: All Format: U10 zero-based DWord offset into the HW status page. Address: HardwareStatusPageOffset[11:2] Surface Type: U32 Range [16, 1023] This field specifies the offset (into the hardware status page) to which the data will be written. Note that the first few DWords of this status page are reserved for special-purpose data storage – targeting these reserved locations via this command is UNDEFINED.
	1:0	**Reserved** Project: All Format: MBZ
2	31:0	**Data DWord 0** Project: All Format: U32 This field specifies the DWord value to be written to the targeted location. For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).
3	31:0	**Data DWord 1** Project: All Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

10.6 MI_USER_INTERRUPT

MI_USER_INTERRUPT			
Project:	All	**Length Bias:**	1
The MI_USER_INTERRUPT command is used to generate a User Interrupt condition. The parser will continue parsing after processing this command. See User Interrupt.			

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 02h MI_USER_INTERRUPT Format: OpCode
	22:0	**Reserved** Project: All Format: MBZ

10.7 MI_WAIT_FOR_EVENT

MI_WAIT_FOR_EVENT			
Project:	All	**Length Bias:**	1

The MI_WAIT_FOR_EVENT command is used to pause command stream processing until a specific event occurs or while a specific condition exists. See Wait Events/Conditions, Device Programming Interface in *MI Functions*. Only one event/condition can be specified -- specifying multiple events is UNDEFINED.

The effect of the wait operation depends on the source of the command. If executed from a batch buffer, the parser will halt (and suspend command arbitration) until the event/condition occurs. If executed from a ring buffer, further processing of that ring will be suspended, although command arbitration (from other rings) will continue. Note that if a specified condition does not exist (the condition code is inactive) at the time the parser executes this command, the parser proceeds, treating this command as a no-operation.

If execution of this command from a primary ring buffer causes a wait to occur, the active ring buffer will *effectively* give up the remainder of its time slice (required in order to enable arbitration from other primary ring buffers).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 03h MI_WAIT_FOR_EVENT Format: OpCode
	22:0	**Reserved** Project: All Format: MBZ

§§

11 Graphics Memory Interface Functions

11.1 Introduction

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

Information not presented in this chapter includes:

- Microarchitectural and implementation-dependent features (e.g., internal buffering, caching and arbitration policies).
- MI functions and paths specific to the operation of external (discrete) devices attached via external connections.
- MI functions essentially unrelated to the operation of the internal graphics devices, e.g., traditional "chipset functions" (refer to the device's C-Spec for this information).

11.2 Graphics Memory Clients

The MI function provides memory access functionality to a number of external and internal graphics memory *clients*, as described in Table 11-1.

Table 11-1. Graphics Memory Clients

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

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

11.3 Graphics Memory Addressing Overview

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

11.3.1 Graphics Address Path

Figure 11-1 shows the internal graphics memory address path, connection points, and optional operations performed on addresses. Externally-supplied addresses are normalized to zero-based *Graphics Memory* (*GM) addresses* (GM_Address). If the GM address is determined to be a tiled address (based on inclusion in a fenced region or via explicit surface parameters), *address tiling* is performed. At this point the address is considered a *Logical Memory address*, and is translated into a *Physical Memory address* via the GTT and associated TLBs. The physical memory location is then accessed.

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

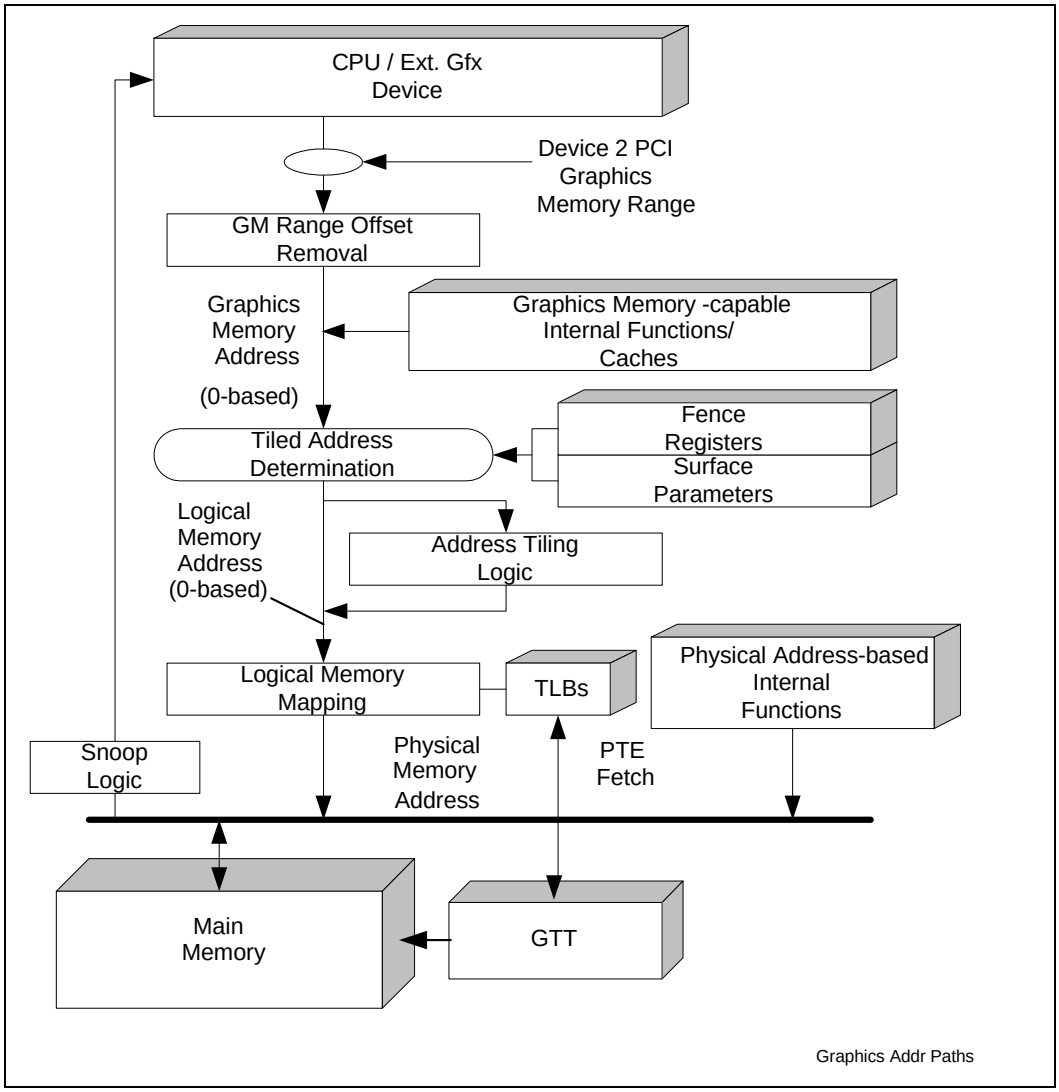


Figure 11-1. Graphics Memory Paths

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

11.4 Graphics Memory Address Spaces

Table 11-2 lists the five supported Graphics Memory Address Spaces. Note that the Graphics Memory Range Removal function is automatically performed to transform system addresses to internal, zero-based Graphics Addresses.

Table 11-2. Graphics Memory Address Types

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

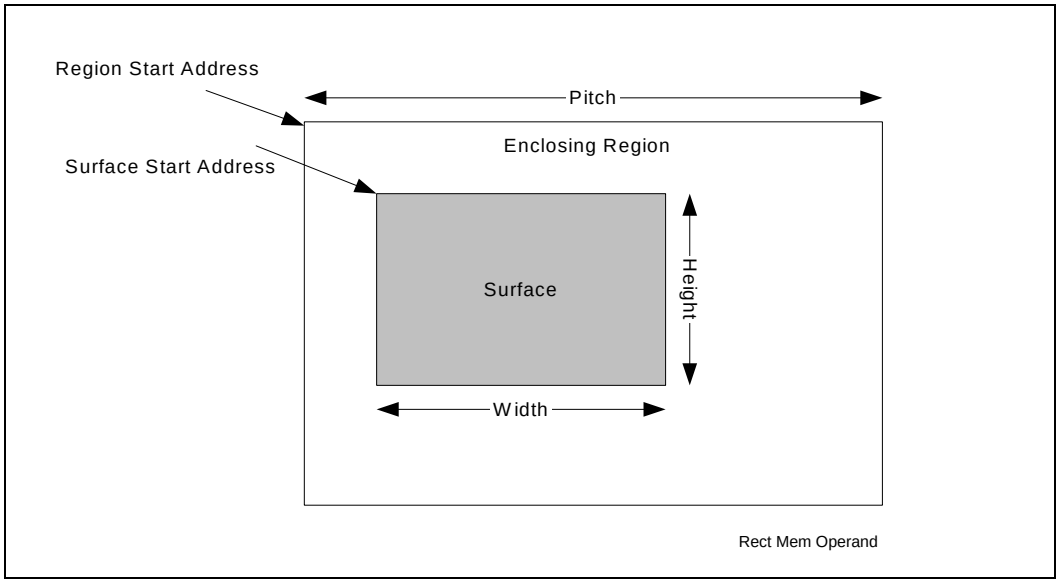
11.5 Address Tiling Function

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

11.5.1 Linear vs. Tiled Storage

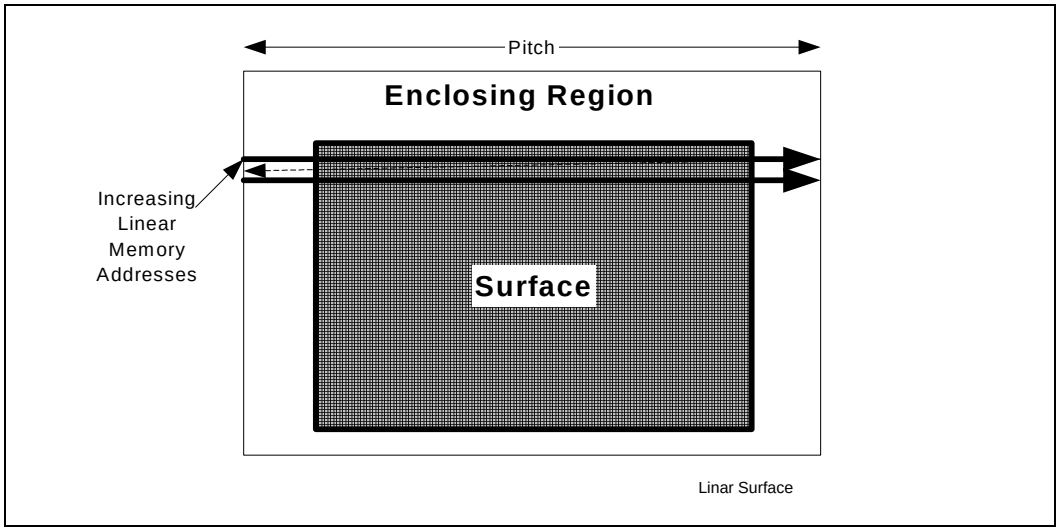
Regardless of the memory storage format, “rectangular” memory operands have a specific *width* and *height*, and are considered as residing within an enclosing rectangular region whose width is considered the *pitch* of the region and surfaces contained within. Surfaces stored within an enclosing region must have widths less than or equal to the region pitch (indeed the enclosing region may coincide exactly with the surface). Figure 11-2 shows these parameters.

Figure 11-2. Rectangular Memory Operand Parameters



The simplest storage format is the *linear* format (see Figure 11-3), where each row of the operand is stored in sequentially increasing memory locations. If the surface width is less than the enclosing region's pitch, there will be additional memory storage between rows to accommodate the region's pitch. The pitch of the enclosing region determines the distance (in the memory address space) between vertically-adjacent operand elements (e.g., pixels, texels).

Figure 11-3. Linear Surface Layout



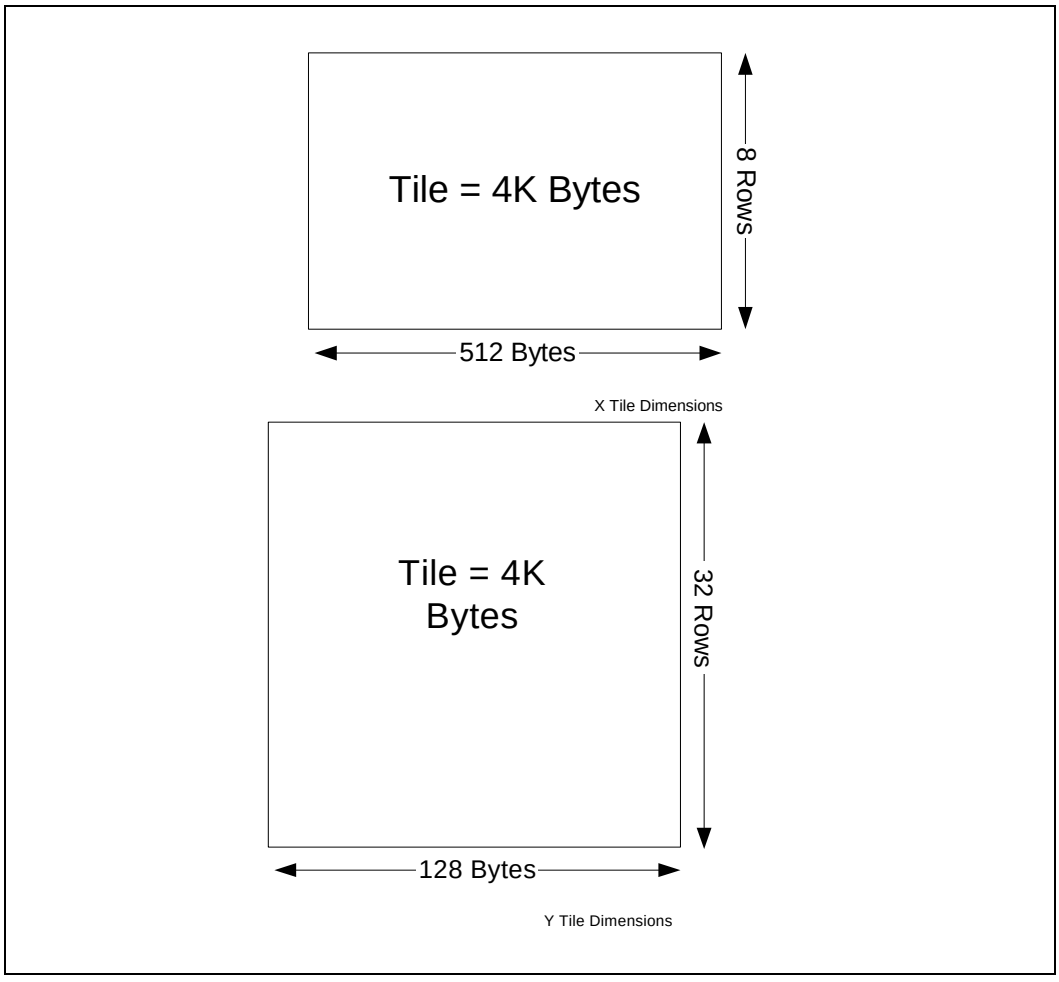
The linear format is best suited for 1-dimensional row-sequential access patterns (e.g., a display surface where each scanline is read sequentially). Here the fact that one object element may reside in a different memory page than its vertically-adjacent neighbors is not significant; all that matters is that horizontally-adjacent elements are stored contiguously. However, when a device function needs to access a 2D subregion

within an operand (e.g., a read or write of a 4x4 pixel span by the 3D renderer, a read of a 2x2 texel block for bilinear filtering), having vertically-adjacent elements fall within different memory pages is to be avoided, as the page crossings required to complete the access typically incur increased memory latencies (and therefore lower performance).

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

Tiles have a fixed 4KB size and are aligned to physical DRAM page boundaries. They are either 8 rows high by 512 bytes wide or 32 rows high by 128 bytes wide (see Figure 11-4). Note that the dimensions of tiles are irrespective of the data contained within – e.g., a tile can hold twice as many 16-bit pixels (256 pixels/row x 8 rows = 2K pixels) than 32-bit pixels (128 pixels/row x 8 rows = 1K pixels).

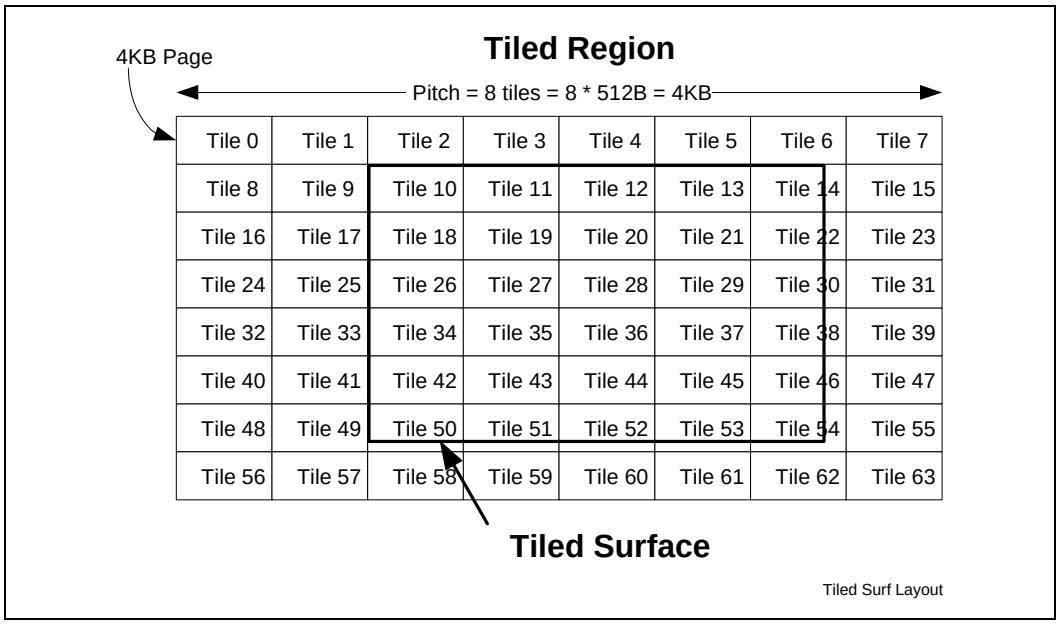
Figure 11-4. Memory Tile Dimensions



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

Figure 11-5 shows an example of a tiled surface located within a tiled region with a pitch of 8 tile widths (512 bytes * 8 = 4KB). Note that it is the enclosing region that is divided into tiles – the surface is not necessarily aligned or dimensioned to tile boundaries.

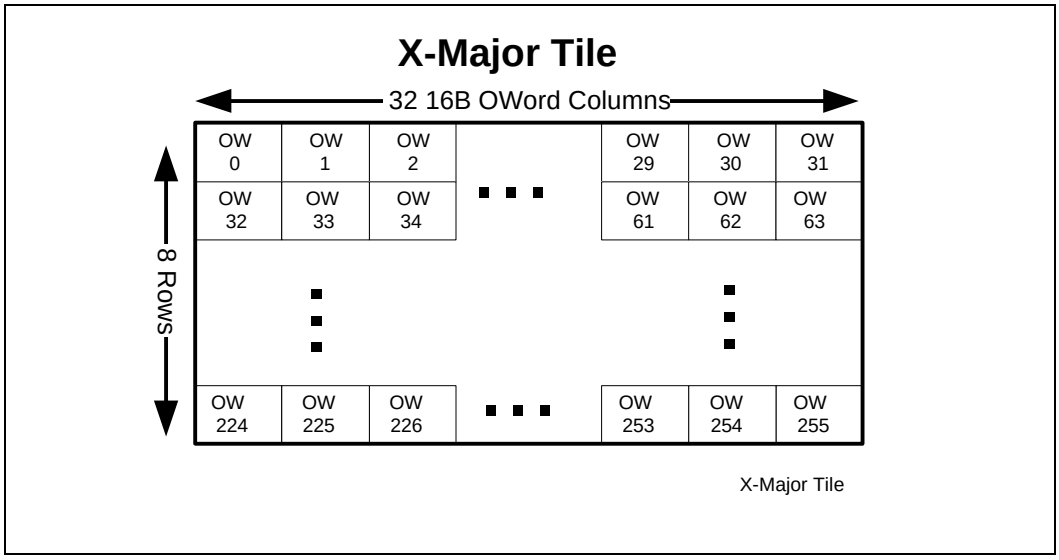
Figure 11-5. Tiled Surface Layout



11.5.2 Tile Formats

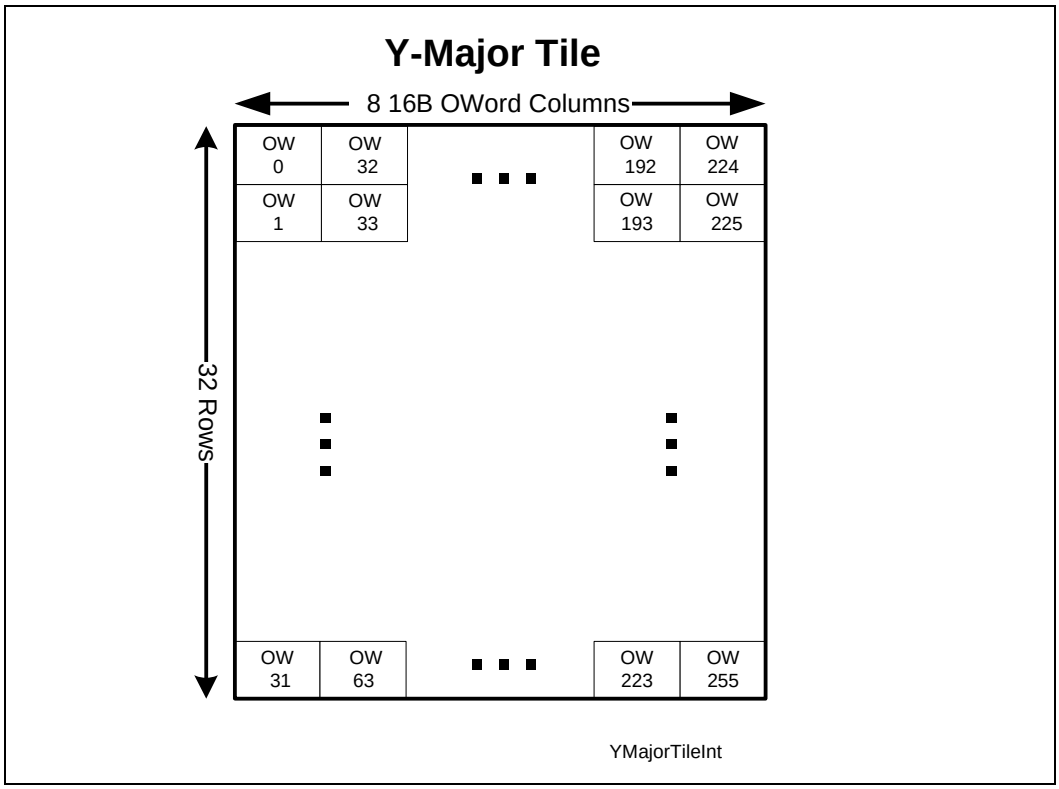
The device supports both *X-Major* (row-major) and *Y-Major* (column major) storage of tile data units, as shown in the following figures. A 4KB tile is subdivided into an 8-high by 32-wide array of 16-byte OWords for X-Major Tiles (X Tiles for short), and 32-high by 8-wide array of OWords for Y-Major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

Table 11-3. X-Major Tile Layout



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

Figure 11-6. Y-Major Tile Layout



11.5.3 Tiling Algorithm

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

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

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

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

If (Fenced) {
       LinearAddress = LinearAddress – FenceBaseAddress
       LinearAddrInTileW = LinearAddress div TileW;
       Xoffset_inTile = LinearAddress mod TileW;
       Y = LinearAddrInTileW div Pitch;
       X = LinearAddrInTileW mod Pitch + Xoffset_inTile;
}

// Internal graphics clients that access tiled memory already have
the X, Y
// coordinates and can start here
YOff_Within_Tile = Y mod TileH;
XOff_Within_Tile = X mod TileW;

TileNumber_InY = Y div TileH;
TileNumber_InX = X div TileW;

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

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

TiledOffset = WalkY? TiledOffsetY : TiledOffsetX;

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

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

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

11.5.4 Tiling Support

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

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

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

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

11.5.4.1 Tiled (Fenced) Regions

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

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

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

11.5.4.2 Tiled Surface Parameters

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

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

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

11.5.4.3 Tiled Surface Restrictions

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

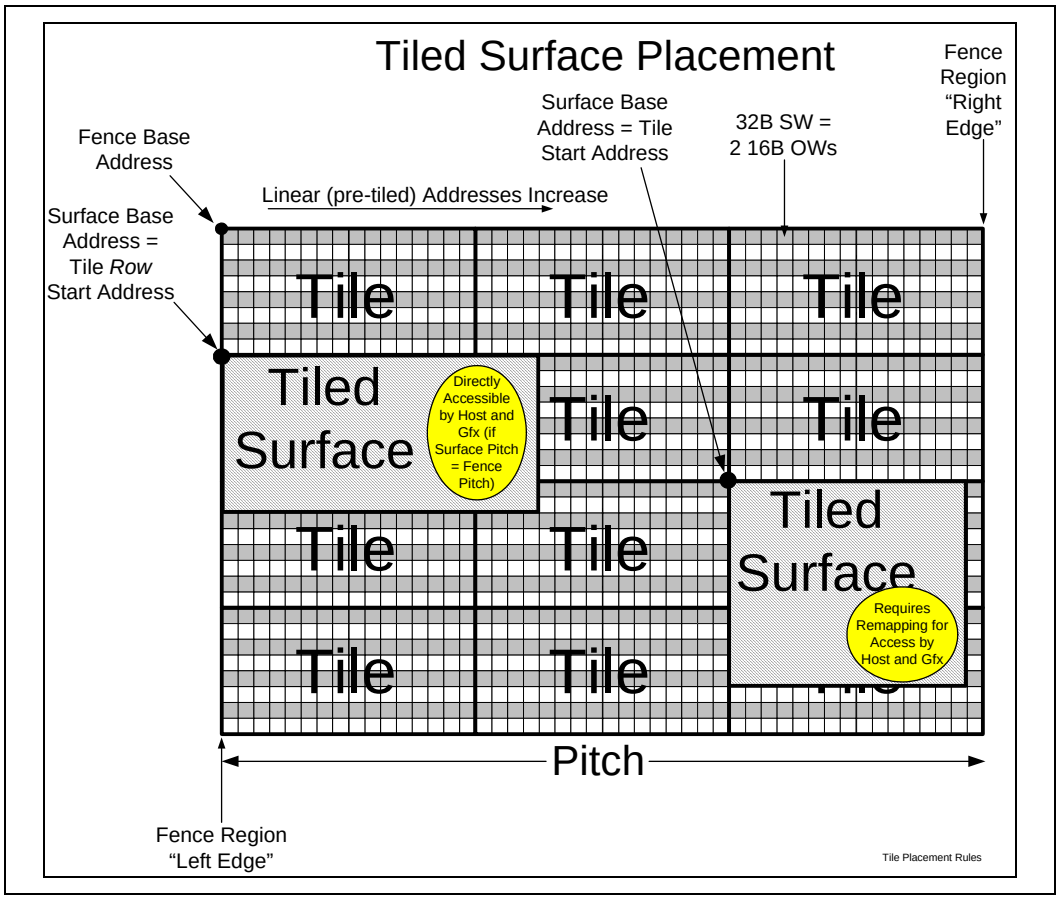
The tiling algorithm for internal device functions is different from that of fence regions. Internal device functions always specify tiling in terms of a surface. The surface must have a base address, and this base address is not subject to the tiling algorithm. Only *offsets* from the base address (as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

Fence regions have no base address per se. Host linear addresses that fall in a fence region are translated in their entirety by the tiling algorithm. It is as if the surface being tiled by the fence region has a base address in graphics memory equal to the fence base address, and all accesses of the surfaces are (possibly quite large) offsets from the fence base address. Fence regions have a virtual “left edge” aligned with the

fence base address, and a “right edge” that results from adding the fence pitch to the “left edge”. Surfaces in the fence region must not straddle these boundaries.

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

Figure 11-7. Tiled Surface Placement



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

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

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

11.5.5 Per-Stream Tile Format Support

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

11.6 Logical Memory Mapping

In order to provide a contiguous address space for graphics operands (surfaces, etc.) yet allow this address space to be mapped onto possibly discontiguous physical memory pages, the internal graphics device supports a Logical Memory Space (see Figure 11-10). A global *Graphics Translation Table* (GTT) is provided to map zero-based (and post-tiled) Logical Memory Addresses into a set of 4KB physical memory pages. (This mapping is also used for external PEG devices.)

There is another logical mapping function available local to each graphics process; this works identically to the global GTT with some additional restrictions. The base address for this per-process GTT (PPGTT) is determined by the PGTBL_CTL2 register. This register is saved and restored with ring context, thus providing each graphics context with its own local translation table and protected memory space (see Rendering Context Management later in this chapter).

The GTT and PPGTT are arrays of 4-byte *Page Table Entries* (PTEs) physically located in Main Memory. The GTT and PPGTT are comprised of a number of locked (non-

swappable) physically-contiguous 4KB memory pages, with a maximum size (each) of 128 4KB pages (128K DWords map 128K*4KB = 512MB max) for Global GTT, and up to 512 4KB pages for PPGTT, for total up to 2GB max. GTT and PPGTT base addresses must be 4KB-aligned.

Note that the PTEs within the global GTT must be written only through GTTADDR (see the Device #2 Config registers for a description of this range), as the MI function needs to snoop PTE updates in order to invalidate TLBs, which cache PTEs. The PGTBL_CTL register also contains a Page Table Enable bit used to enable/disable Logical Memory mapping. With the exception of processor Read, Cursor and VGA clients, access to graphics memory is not permitted when the Page Table Enable bit is clear (i.e., disabled). The PGTBL_ER debug register provides information pertaining to HW-detected errors in the Logical Memory Mapping function (e.g., invalid PTEs, invalid mappings, etc.).

The PPGTT base address is also 4KB aligned, but it is programmed directly in physical memory space rather than through an alias mechanism like GTTADDR. Note that not all clients may use the PPGTT; only the global GTT is available for processor accesses as well as graphics accesses from display engines (including overlay and cursor). Any per-process access that occurs while the PPGTT is disabled (via a bit in PGTBL_CTL2) will default to a translation via the global GTT.

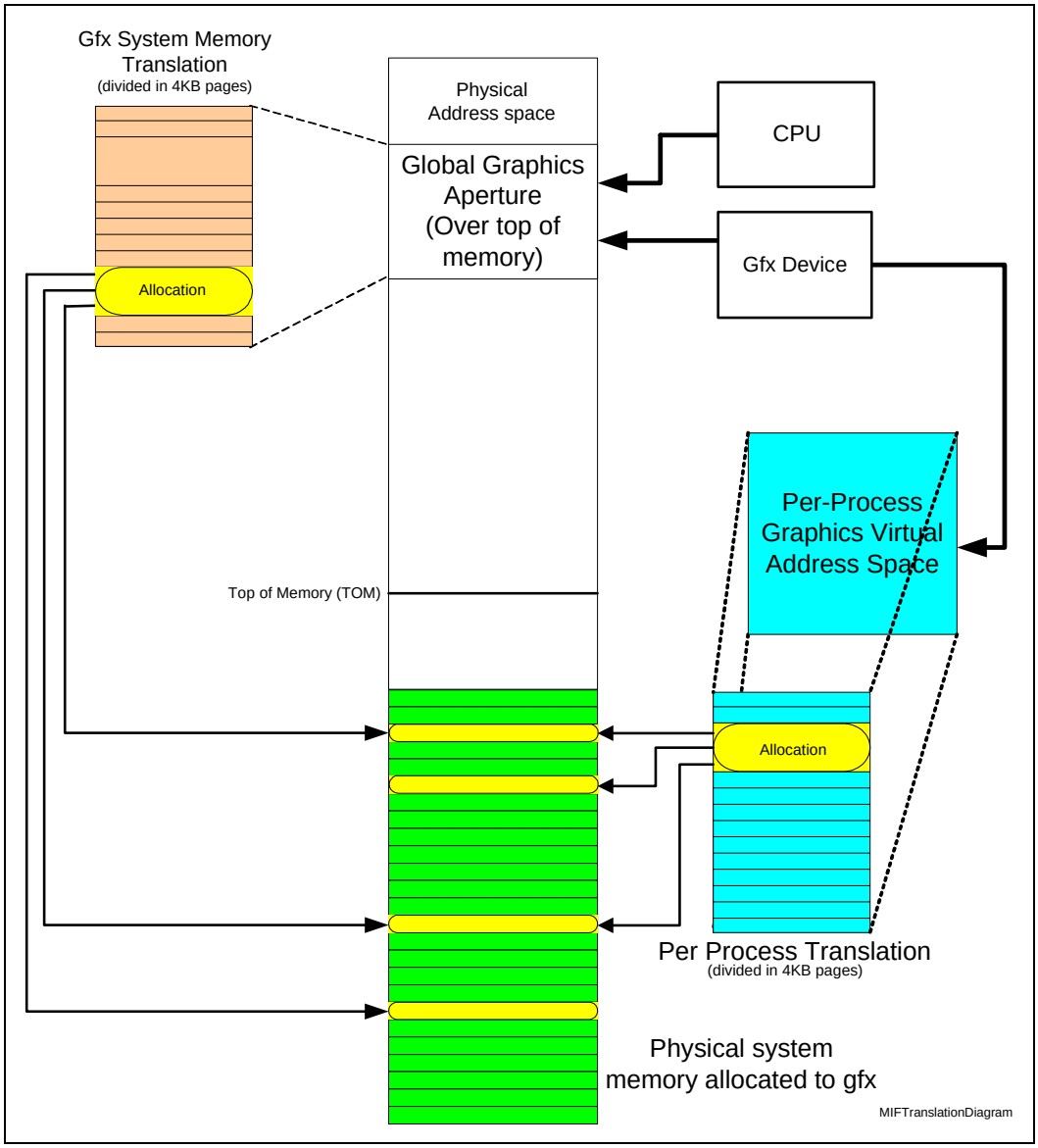
11.6.1 Logical Memory Space Mappings

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

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

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

Figure 11-8. Global and Render GTT Mapping



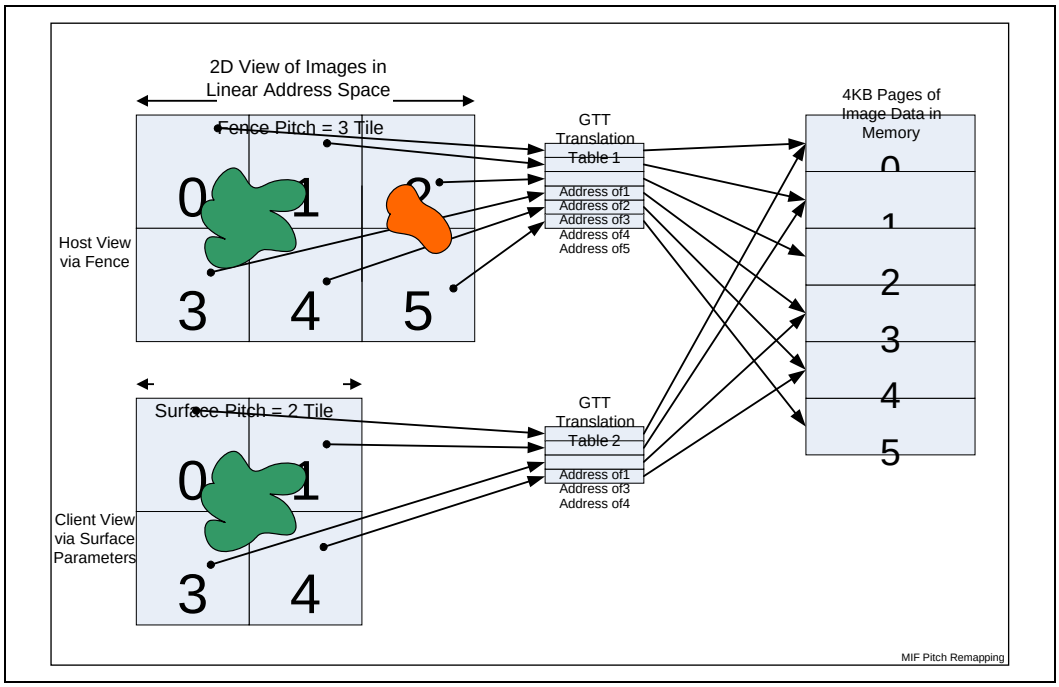
The following table lists the memory space mappings valid for each MI client:

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

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

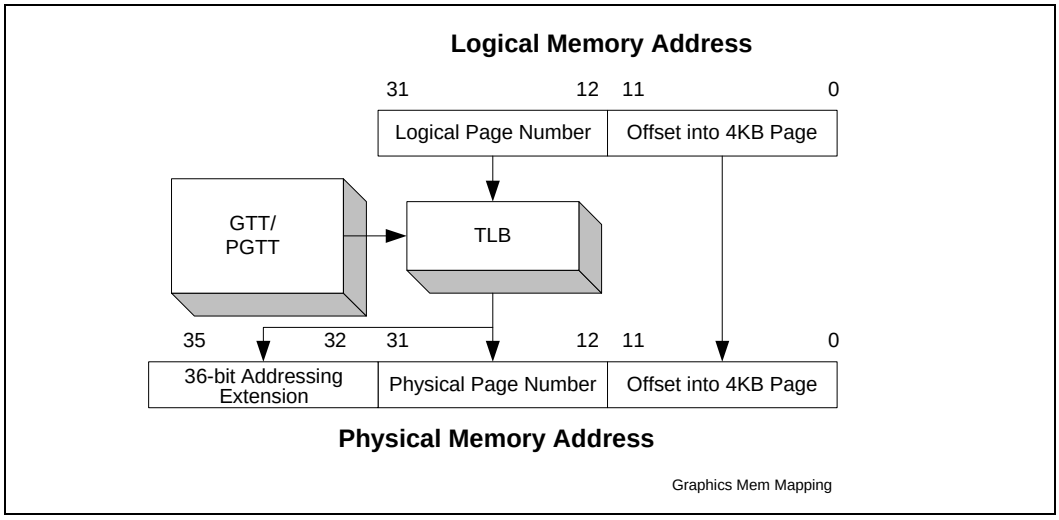
The PPGTT mechanism can be used by a client to access a surface with a pitch that is smaller than that of the fence region used by the host to initialize the surface, without having to physically move the data in memory.

Figure 11-9. GTT Re-mapping to Handle Differing Pitches



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

Figure 11-10. Logical-to-Physical Graphics Memory Mapping

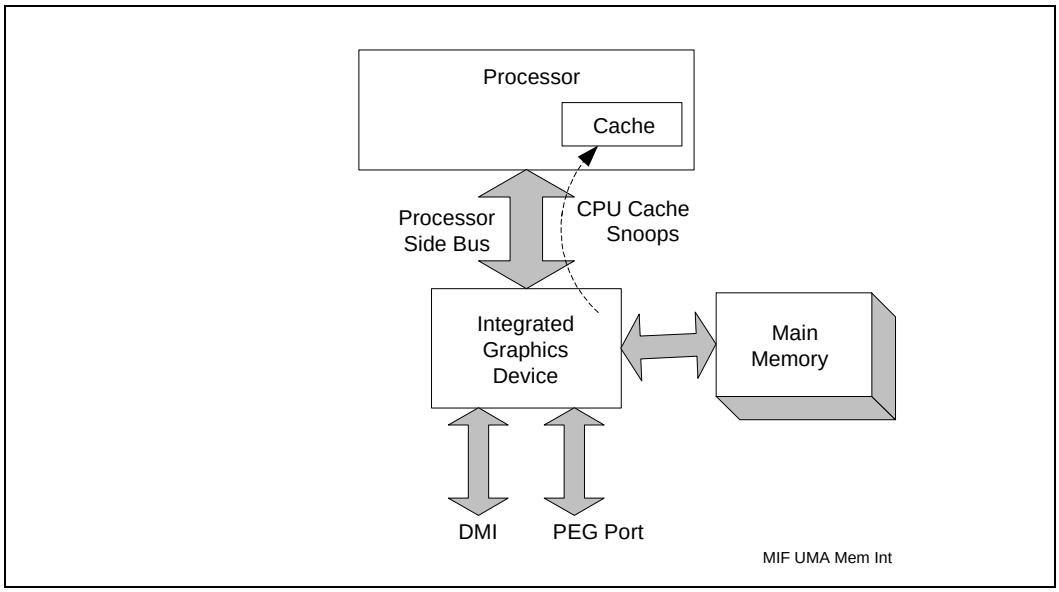


11.7 Physical Graphics Memory

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

Figure 11-11 shows how the Main Memory is interfaced to the device.

Figure 11-11. Memory Interfaces



11.7.1 Physical Graphics Address Types

Table 11-4 lists the various physical address types supported by the integrated graphics device. Physical Graphics Addresses are either generated by Logical Memory mappings or are directly specified by graphics device functions. These physical addresses are not subject to tiling or GTT re-mappings.

Table 11-4. Physical Memory Address Types

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

11.7.2 Main Memory

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

11.7.2.1 Optimizing Main Memory Allocation

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

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

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

11.7.2.2 Application of the Theory (Page Coloring)

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

11.7.2.2.1 3D Color and Depth Buffers

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

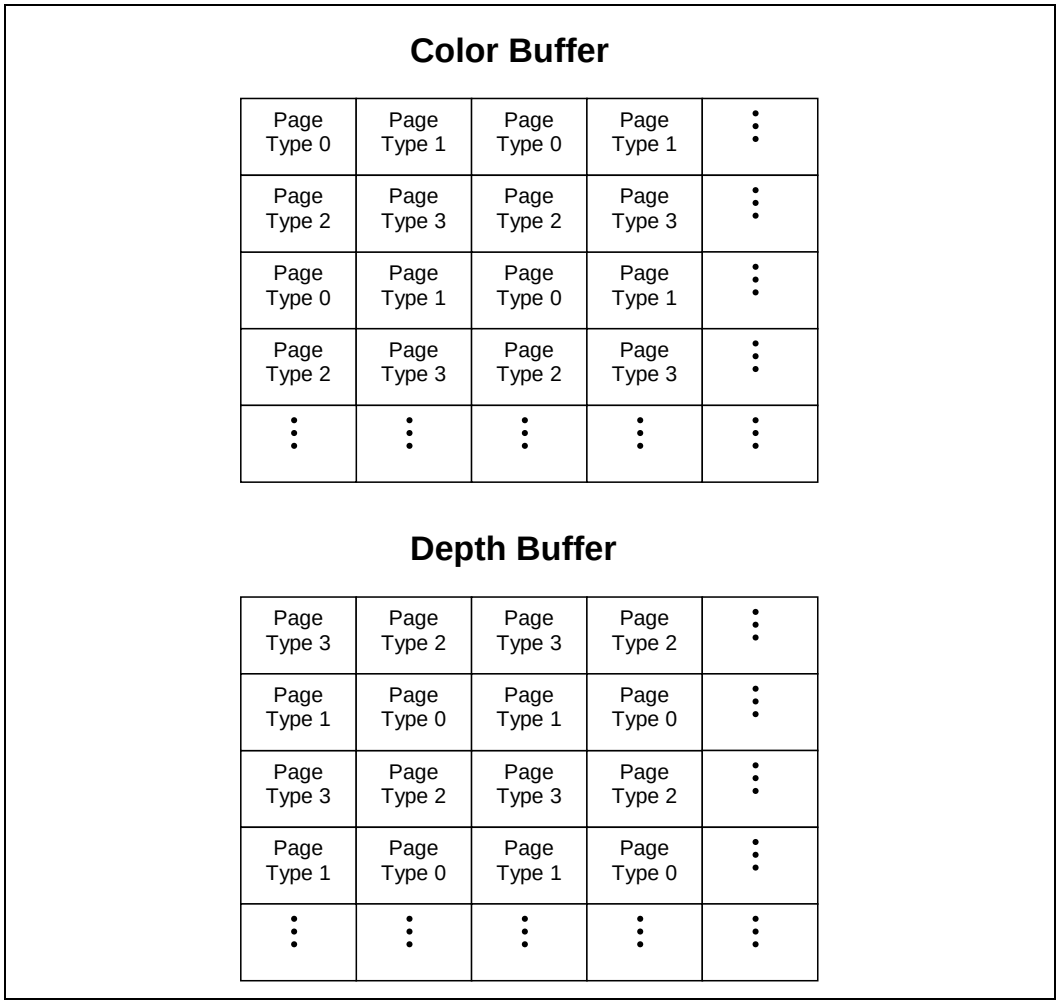


Figure 11-12. Memory Pages backing Color and Depth Buffers

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

11.7.2.2.2 Media / Video

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

§§

12 Device Programming Environment

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

12.1 Programming Model

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

POST-Time Programming of Configuration Registers

These registers are the graphics device registers residing in PCI space. A majority of these registers are programmed once during POST of the video device. Configuration registers are not covered in this section. For details on accessing the graphics device’s configuration space see the EDS.

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

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

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

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

12.2 Graphics Device Register Programming

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

Programming Note:

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

12.3 Graphics Device Command Streams

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

12.3.1 Command Use

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

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

12.3.2 Command Transport Overview

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

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

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

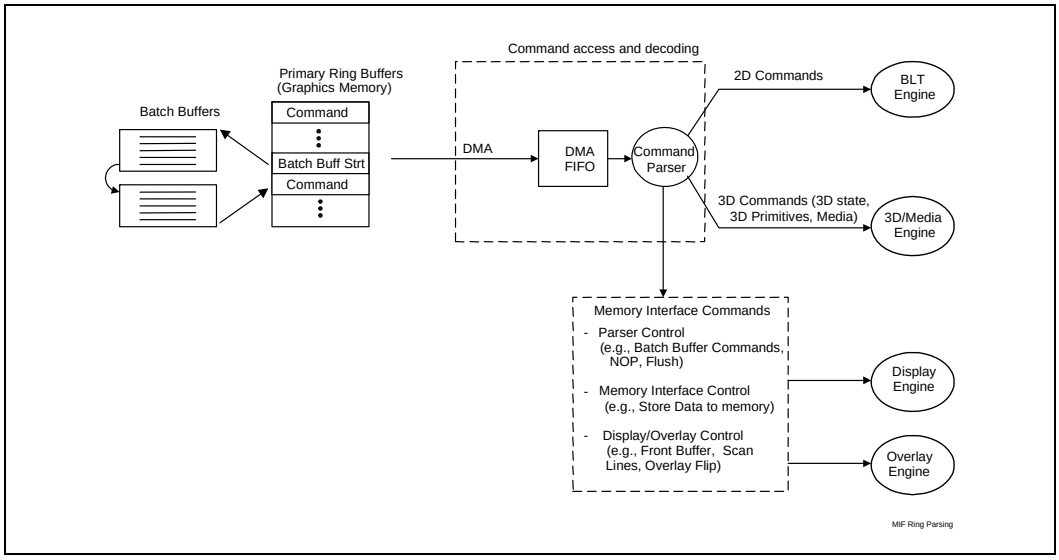
12.3.3 Command Parser

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

- Detecting the presence of commands (within the Ring Buffer).
- Reading commands from the Ring Buffer and Batch Buffers via DMA. This includes support of the automatic head report function.
- Parsing the common "Command Type" (destination) field of commands.
- Execution of Memory Interface commands that control CP functionality, provide synchronization functions, and provide display and overlay flips as well as other miscellaneous control functions.
- Redirection of 2D, 3D and Media commands to the appropriate destination (as qualified by the INSTPM register) while enforcing drawing engine concurrency and coherency rules.
- Performing the "Sync Flush" mechanism
- Enforcing the Batch Buffer protection mechanism

Figure 12-1 is a high-level diagram of the graphics device command interface.

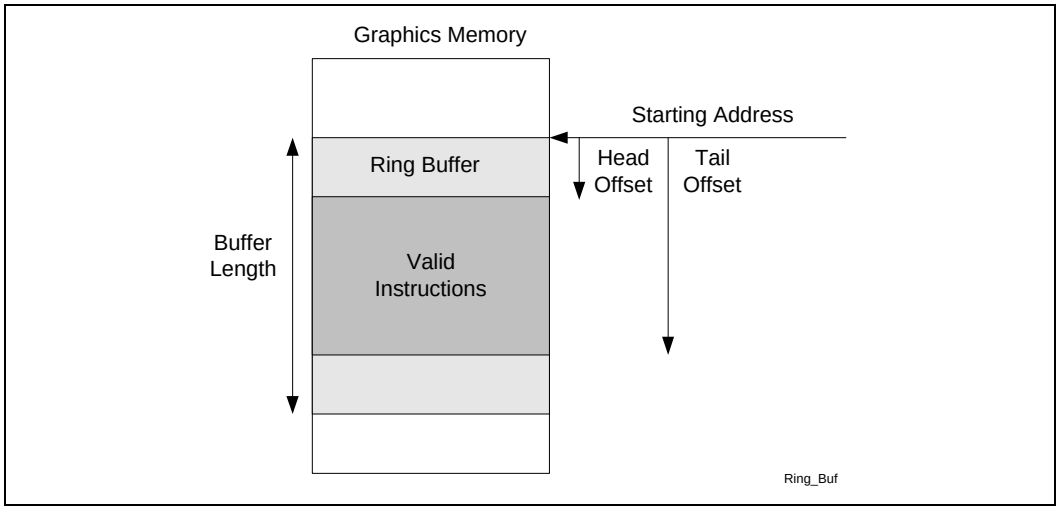
Figure 12-1. Graphics Controller Command Interface



12.3.4 The Ring Buffer

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

Figure 12-2. Ring Buffer



12.3.4.1 The Ring Buffer (RB)

Ring Buffer support:

- Batch Buffer initiation
- Indirect Data (operand access)

12.3.4.2 Ring Buffer Registers

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

- **Ring Buffer Valid:** This bit controls whether the Ring Buffer is included in the command arbitration process. Software must program all other Ring Buffer parameters before enabling a Ring Buffer. Although a Ring Buffer can be enabled in the non-empty state, it must not be disabled unless it is empty. Attempting to disable a Ring Buffer in the non-empty state is UNDEFINED. Enabling or disabling a Ring Buffer does not of itself change any other Ring Buffer register fields.
- **Start Address:** This field points to a contiguous, 4KB-aligned, linear (i.e., must not be tiled), mapped graphics memory region which provides the actual command buffer area. Writing the Start Address has the side effect of clearing the Head Offset and Head Wrap Count fields.
- **Buffer Length:** The size of the buffer, in 4KB increments, up to 2MB.
- **Head Offset:** This is the DWord offset (from Start Address) of the next command that the CP will parse (i.e., it points one DWord past the last command parsed). The CP will update this field as commands are parsed – the CP typically continues parsing new commands before the previous command operations complete. (Note that, if commands are pending execution, the CP will likely have prefetched commands past the Head Offset). As the graphics device does not "reset" the Head Offset when a Ring Buffer is enabled, software must program the Head Offset field before enabling the Ring Buffer. Software can enable a Ring Buffer with any legal values for Head/Tail (i.e., can enable the Ring Buffer in an non-empty state). It is anticipated, but not required, that software enable The Ring

Buffer with Head and Tail Offsets of 0. Once the Head Offset reaches the QWord specified by the Tail Offset (i.e., the offsets are equal), the CP considers the Ring Buffer "empty".

- **Head Wrap Count:** This field is incremented by the CP every time the Head Offset wraps back to the start of the buffer. As it is included in the DWord written in the "report head" process, software can use this field to track CP progress as if the Ring Buffer had a "virtual" length of 2048 times the size of the actual physical buffer (up to 4GB).
- **Tail Offset:** This is the offset (from Start Address) of the next QWord of command data that software will request to be executed (i.e., it points one DWord past the last command DWord submitted for execution). The Tail Offset can only point to an command boundary – submitting partial commands is UNDEFINED. As the Tail Offset is a QWord offset, this requires software to submit commands in multiples of QWords (both DWords of the last QWord submitted must contain valid command data). Software may therefore need to insert a “pad” command to meet this restriction. After writing commands into the Ring Buffer, software updates the Tail Offset field in order to submit the commands for execution (by setting it to the QWord offset past the last command). The commands submitted can wrap from the end of the buffer back to the top, in which case the Tail Offset written will be less than the previous value. As the “empty” condition is defined as “Head Offset == Tail Offset”, the largest amount of data that can be submitted at any one time is one QWord less than the Ring Buffer length.
- **IN USE Semaphore Bit:** This bit (included in the Tail Pointer register) is used to provide a HW semaphore that SW can use to manage access to the individual The Ring Buffer. See the Ring Buffer Semaphore section below.
- **Automatic Report Head Enable:** Software can request to have the hardware Head Pointer register contents written ("reported") to snooped system memory on a periodic basis. Auto-reports can be programmed to occur whenever the Head Offset crosses either a 64KB or 128KB boundary. (Note therefore that a Ring Buffer must be at least 64KB in length for the auto-report mechanism to be useful). The complete Head Pointer register will be stored at a Ring Buffer-specific DWord offset into the "hardware status page" (defined by the HWSTAM register). The auto-report mechanism is desirable as software needs to use the Head Offset to determine the amount of free space in the Ring Buffer -- and having the Head Pointer periodically reported to system memory provides a fairly up-to-date Head Offset value automatically (i.e., without having to explicitly store a Head Pointer value via the MI_REPORT_HEAD command).

Table 12-1. Ring Buffer Characteristics

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

12.3.4.3 Ring Buffer Placement

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

12.3.4.4 Ring Buffer Initialization

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

12.3.4.5 Ring Buffer Use

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

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

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

1. A direct read (poll) of the Head Pointer register. This gives the most accurate indication but is expensive due to the uncached read.
2. The automatic reporting of the Head Pointer register in the Hardware Status Page. This has low impact as no uncached reads or command overhead is involved. However, given the 64KB/128KB granularity of auto-reports, this mechanism only works well on fairly large The Ring Buffer.
3. The explicit reporting of the Head Pointer register via the MI_REPORT_HEAD command. This allows for flexible and more accurate reporting but comes at the cost of command bandwidth and execution time, in addition to the software overhead to determine how often to report the head.
4. Some other "implicit" means by which software can determine how far the CP has progressed in retiring commands from a Ring Buffer. This could include the use of "Store DWORD" commands to write sequencing data to system memory. This has similar characteristics to using the MI_REPORT_HEAD mechanism.

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

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

12.3.4.6 Ring Buffer Semaphore

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

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

12.3.5 Batch Buffers

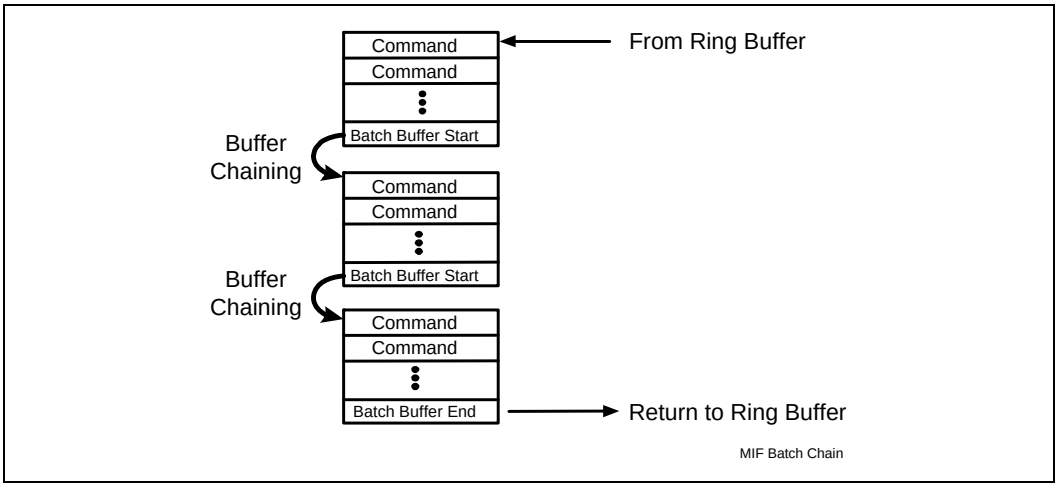
The graphics device provides for the execution of command sequences *external* to the Ring buffer. These sequences are called "Batch Buffers", and are initiated through the use of various Batch Buffer commands described below. When a Batch Buffer command is executed, a batch buffer sequence is initiated,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

12.3.5.1 Batch Buffer Chaining

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

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

Figure 12-3. Batch Buffer Chaining



12.3.5.2 Ending Batch Buffers

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

12.3.6 Indirect Data

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

12.3.6.1 Logical Contexts

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

12.3.7 Command Arbitration

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

12.3.7.1 Arbitration Policies and Rationale

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

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

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

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

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

12.3.7.2 Wait Commands

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

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

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

12.3.7.3 Wait Events/Conditions

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

12.3.7.3.1 Display Pipe A,B Vertical Blank Event

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

12.3.7.3.2 Display Pipe A,B Horizontal Blank Event

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

12.3.7.3.3 Display Plane A, B, C , Flip Pending Condition

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

12.3.7.3.4 Overlay Flip Pending Condition

The Overlay Flip Pending condition is similar to the Display Flip Pending condition, with the exception that overlay flips are only performed synchronously with display refresh.

12.3.7.3.5 Display Pipe A,B Scan Line Window Conditions

The graphics device supports two conditions relating to the progress of refresh within a particular display stream. A “Scan Line Window” is defined as the period of time between the refresh of two specific display scan lines. The MI_WAIT_ON_EVENT command can be used to pause an command stream while a particular display refresh is inside or outside the Scan Line Window. (Actually, the MI_WAIT_EVENT command only supports waiting on the Scan Line Window condition, and the MI_LOAD_SCAN_LINES_INCL or MI_LOAD_SCAN_LINES_EXCL are used to define an “inclusive” or “exclusive” window).

If no Scan Line Window has been defined for the particular display stream, the MI_WAIT_EVENT specifying the Scan Line Window event will never introduce a wait.

12.3.7.3.6 Semaphore Wait Condition

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

12.3.7.4 Command Arbitration Points

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

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

12.3.7.5 Command Arbitration Rules

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

12.3.7.6 Batch Buffer Protection

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

This protection mechanism is enabled via a field in a Batch Buffer command that indicates whether the associated batch buffer is “secure” or “non-secure”. When the

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

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

12.3.8 Graphics Engine Synchronization

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

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

12.3.9 Graphics Memory Coherency

Table 12-2 lists the various types of graphics memory coherency provided by the device, specifically where the CPU writes to a 64B cacheline, and the device then accesses that same cacheline. Note that the coherency policy depends on the address type (GM or MM) involved in the accesses.

Table 12-2. Graphics Memory Coherency

CPU Access	Subsequent Device Access	Example Operand	Coherency
Write GM	Read GM		TBD
Write MM	Read MM	Batch Buffer	TBD
Write GM	Write GM		Device ensures coherency following every Ring Buffer Tail Pointer write. (This can be made optional via a bit in the Tail Pointer data).
Write MM	Write MM		TBD "assumed to exclusive byte" ?
Write GM	Read MM		Device ensures coherency following every Ring Buffer Tail Pointer write. (This can be made optional via a bit in the Tail Pointer data).

12.3.10 Graphics Cache Coherency

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

12.3.10.1 Rendering Cache

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

- Execution of an MI_FLUSH command with the **Render Flush Cache Inhibit** bit clear
- Execution of a PIPE_CONTROL instruction with the **Write Cache Flush Enable** bit set (Depth Stall must be clear).
- A SyncFlush handshake operation
- A change of rendering engines (e.g., going from 2D to 3D, 3D to 2D, etc.)
- Logical Context switch (via MI_SET_CONTEXT)
  - The render cache must be explicitly flushed using one of these mechanisms under certain conditions. See Graphics Engine Synchronization above.

12.3.10.2 Sampler Cache

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

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

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

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

12.3.10.3 Instruction/State Cache

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

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

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

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

12.3.10.4 Vertex Cache

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

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

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

12.3.10.5 GTT TLBs

The following table summarizes when the various TLBs are invalidated.

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

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

12.3.11 Command Synchronization

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

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

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

Given the CP rules for command execution, it is possible to use the indication of command parsing progress to infer the retirement status of parsed commands. The

only indication of instruction retirement available from instruction parsing is that parsing of an MI instruction implies retirement of previous MI instructions with the following exceptions:

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

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

12.3.11.1 MI_FLUSH

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

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

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

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

See section 12.3.10.2 for more information on when the sampler cache should be invalidated.

12.3.11.2 Sync Flush

Inserting MI_FLUSH commands, while effective at determining or forcing the retirement of previous rendering commands, may negatively impact performance if not absolutely required. For example, if the knowledge of rendering command retirement is not known a priori, it is likely undesirable to insert MI_FLUSH commands at intervals in the command stream. However, it may not be acceptable to insert an MI_FLUSH command (and wait for its completion) at the point that rendering command retirement is required – as there may be a large number of commands pending in ring/batch buffers at that point and flushing the entire device (including waiting for completion of pending commands that have not yet been parsed) may be prohibitive. There is a mechanism, however, where command stream synchronization

can be performed on demand, without requiring earlier submitted commands and batch buffers to complete – it is called the “Sync Flush” mechanism.

Here’s how it works:

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

12.4 Hardware Status

The graphics device supports a number of internal hardware status bits which can be used to detect and monitor hardware status conditions via polling or interrupts. This section will describe each hardware status bit. The following section describes the hardware status reporting (polling) mechanism. The mechanism to allow these status bits to generate interrupts is described in the Interrupts section. Note that the hardware status bits are actually reported in the Interrupt Status Register, so “hardware status” and “interrupt status” are used interchangeably here (though many hardware status bits won’t necessarily ever be used to generate interrupts).

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

12.4.1 Hardware-Detected Errors (Master Error bit)

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

12.4.2 Thermal Sensor Event

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

12.4.3 Sync Status

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

12.4.4 Display Plane A, B, Flip Pending

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

12.4.5 Overlay Flip Pending

This bit is similar to the Display Flip Pending bits. It is set to ‘1’ when the MI_OVERLAY_FLIP command is executed. It is cleared to ‘0’ after the overlay registers are read from memory during the next vertical blanking period.

12.4.6 Display Pipe A,B VBLANK

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

12.4.7 User Interrupt

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

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

12.4.8 PIPE_CONTROL Notify Interrupt

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

12.4.9 Display Port Interrupt

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

12.5 Hardware Status Writes

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

12.6 Interrupts

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

12.7 Errors

The graphics device supports the hardware detection of a number of *operational* and *debug-only* errors. Operational errors occur out of the immediate control of driver software and must be anticipated and tolerated to the extent required by the relevant APIs. Software must therefore support the detection and proper handling of all relevant operational errors. The (more numerous) debug-only errors are just that – detected to facilitate initial system debug but not intended to be tolerated during normal system operation. In many cases, debug-only errors are not recoverable. They require the use of debug registers to detect and diagnose.

12.7.1 Error Reporting

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

There are three registers dedicated to control, detect, and clear hardware error status conditions in a similar fashion to the Interrupt Control Registers. All three error registers share a common error status bit definition.

The Error Status Register (ESR) holds the actual error status bits (each of which may be the logical OR of “source” error bits in various functional registers). The Error Mask Register (EMR) is used to select which error status bit(s) are reported in the Error Identity Register (EIR). The EIR holds the “persistent” values of the unmasked error status bits, and is also used to clear error status conditions. Any bits set in the EIR will raise the Master Error interrupt status condition.

The error conditions corresponding to the error status bits include:

- **Page Table Error** (***Debug only***) – This is a summary of a number of possible errors associated with the mapping function of the GTT. See Table 12-3 for more information.
- **Display or Overlay Underrun** (***Debug only***) – This error is raised when a FIFO underrun condition is detected in the display or overlay isochronous streams. See the description of the Display/Overlay Status Register in the *Display Registers* chapter.
- **Command Error** (***Debug Only***) – This is a summary of a number of command data errors detected by the Command Parser. See Command Errors below for more information.

12.7.2 Page Table Errors

The following tables describe the various sources and types of Page Table Errors. Refer to the description of the PGTBL_ERR register in *Memory Interface Registers* for more details.

Table 12-3. Page Table Error Types

Error	Description	Streams
Invalid GTT PTE	In the process of mapping an address, the MI encountered a GTT PTE that was marked “Invalid”. This would be the result of a programming error.	All (See Table 11-1)
Invalid TLB Miss	An unexpected TLB miss (detected at GTT request time) was encountered (e.g., during Display/Overlay/Sprite access).	Display, Overlay
Invalid PTE Data	Mapping to the physical page specified in the PTE is not permitted (e.g., a page in PAM, SMM or over the top of memory, etc.). This is the result of a programming error.	Host
Invalid Tiling	A tiling parameter was found inconsistent with the current operation. This includes the use of Y-Major tiling in the Render/Display/Overlay streams. This is the result of a programming error. This is detected during GTT request.	Blt, Display, Overlay

NOTE: Note that Page Table Errors cannot be cleared. A device reset is required.

12.7.3 Clearing Errors

For operational errors, software is responsible for taking the proper steps to recover from the error and then clearing the error indication. The actions required to recover from operational errors may be discussed in the various functional areas (not here). See the Hardware-detected Error Bit Definitions in *Memory Interface Registers* for more details. This subsection describes the actions required to clear the error indication.

In order to clear operational errors, software is responsible for clearing the error condition from the source, working back to the Master Error bit. Typically this will entail the following sequence.

- First the primary source of the error must be cleared. This requires clearing the functional register(s) containing the source error indication.
- Next, clear the particular error status bit by writing a ‘1’ to the appropriate bit of the Error Identity Register (EIR). This will clear the error status bit in the Error Status Register (ESR). If multiple errors are present, all error status bits should be cleared simultaneously.
- Next, clear the Master Error interrupt status bit by writing a ‘1’ to the Master Error bit of the Interrupt Identity Register (IIR).

Note: Page Table Errors cannot be cleared.

12.8 Rendering Context Management

The graphics device operation (rendering, etc.) is controlled via the settings of numerous hardware state variables. These state variables are divided into *global state* and *context state*.

There is only one copy of global state variables, and changing the settings of these variables requires explicit programming of the state variables. Examples of global state include:

- MI registers (HWSTAM, Ring Buffer, etc.) with the exception of those listed in the next paragraph (i.e, registers listed there *are* saved/restored)
- Configuration registers
- Display programming registers

On the other hand, context state is associated with a specific *context*, where switching to that context causes that context's state to be restored. While the associated context is active, the state variables and registers can be programmed via the command stream. Examples of context state include the PIPELINE_STATE_POINTERS command and most non-pipelined state. The following MI registers are considered part of context state and thus saved/restored with context:

- INSTPM
- CACHE_MODE_0
- CACHE_MODE_1
- MI_ARB_STATE
- 3D Software Visible Counter Registers

The graphics device supports both a *hardware context* and *logical contexts*. The multiple logical context support provides robust rendering context support by swapping contexts to/from memory.

12.8.1 Multiple Logical Rendering Contexts

The graphics device supports multiple *logical rendering contexts* stored in Main Memory. Logical rendering contexts are referenced via a 2KB-aligned *Logical Context Address*.

The maximum size of a logical context entry (which is information required by the driver to allocate contexts) is currently 2K bytes. For forward compatibility, the maximum size of a logical context entry should be supplied to the drivers via a VBIOS mechanism as opposed to being hardcoded in the driver.

The actual size of a logical rendering context is the amount of data stored/restored during a context switch and is measured in 64B cache lines. There is a debug mechanism that allows software/BIOS to program the actual size of the logical rendering context via the CXT_SIZE register. Note that this register will default to the correct value, so software should not have to modify it.

The format of the logical rendering context in memory is considered device-dependent; software must not attempt to modify the contents of a logical rendering context directly. This restriction is motivated by forward compatibility concerns because the location and definition of fields may change between implementations.

12.8.1.1 Current Context IDs

The ring buffer has an associated *Current Context ID* (CCID) register. The CCID includes a Logical Pipeline Context Address (LPCA).

The CCID for a ring buffer is set during the processing of the new MI_SET_CONTEXT command from that ring. The MI_SET_CONTEXT command provides a new CCID value (LPCA) to be loaded into the CCID register for the associated ring buffer. The MI_SET_CONTEXT command also contains a Restore Inhibit bit used to optionally inhibit the restoration (loading) of the new rendering context. This bit must be used during context initialization to avoid the loading of uninitialized (garbage) context data from memory. Failure to do so leads to UNDEFINED operation.

The initial values of the CCIDs are UNDEFINED. The first time a valid CCID is set from a ring buffer, the normal context save operation will be suppressed, as the previous CCID is invalid.

12.8.1.2 Intra-Ring Context Switch

Within a specific ring buffer, a new logical rendering context is specified via the MI_SET_CONTEXT command. Note that MI_SET_CONTEXT commands are permitted only within a ring buffer (not within a batch buffer).

As part of the execution of the MI_SET_CONTEXT command from within a ring buffer, the Logical Pipeline Context Address fields of the CCID register and MI_SET_CONTEXT command are compared. If they differ (or the CCID register is uninitialized), a rendering context switch operation will be performed, which includes:

1. If the CCID contents are valid, a context save operation will be performed. The contents of the HW context will be saved in memory starting at the Logical Pipeline Context Address specified in the CCID.
2. If the Restore Inhibit command field is not set, a context restore operation will be performed. Here the logical context values are read starting from the Logical Pipeline Context Address field of the command and used to set the internal HW context.
3. The relevant contents of the command will be loaded into the appropriate CCID register. (This occurs irrespective of the LPCA comparison result). At this point, the ring buffer has switched to using the new logical rendering context.

12.8.1.3 Logical Rendering Context Creation and Initialization

12.8.1.3.1 Rendering Context Creation Rules

1. Software only knows the **size** of the logical rendering context (2KB), for allocation purposes.
2. Given (1), software does **not** know the format of the context, and therefore is not allowed to write any portion of a logical rendering context. Software can, however, copy/move entire logical context blocks.
3. Given (2), software must never restore (load) a logical rendering context from memory that has not been previously stored by HW. I.e., software must never attempt to initialize a context itself and then cause it to be loaded. Breaking this rule causes UNDEFINED operation (as in the hang seen in BDG validation).
4. Initialization software must write **all** HW context variables with legal values before the first rendering context can be saved (this must be done before you can perform any rendering anyways). Given this, and the obvious rule that software must never program illegal state values, guarantees that the HW context will forever remain valid (and therefore be available to store into a logical rendering context). Note that software-visible context variables include 3D state, Blt register state, etc.

12.8.1.3.2 Context Initialization

Logical Rendering Contexts can be initialized (in memory) by software in the following way:

1. Issue an MI_SET_CONTEXT command w/ the **Restore Inhibit** bit set and the about-to-be-initialized logical pipeline context address. This will save the current rendering context and then change the LPCA to the new context (without loading it).
2. Use state commands to modify the context as desired.
3. Issue another MI_SET_CONTEXT command specifying some other LPCA (e.g., the previous one). This will cause the new context to be stored (initialized) in memory

12.8.1.4 Context Save

A context save will occur anytime all of the following apply:

- A rendering context switch occurs as a result of the execution of MI_SET_CONTEXT
- the CCID of the current context (CCID register of current ring) and the new CCID (the CCID register of the newly selected ring or the new CCID in the MI_SET_CONTEXT command) differ OR an MI_SET_CONTEXT with the “Force Restore” bit set initiated the context switch
- the current CCID is valid (has been previously set)

The current rendering context will be written out to memory starting at the LPCA in the format described by Logical Context Layout in *Memory Data Formats*. Note that this includes a limited number of Memory Interface Registers whose values are saved by embedding them in an MI_LOAD_REGISTER_IMM command that is written out to memory.

The Optional Extended Context will also be written if the Extended Save Enable bit is set in the current CCID register. Context saves DO NOT modify pipelined state stored in memory.

12.9 Reset State

This section describes the state of the programming interface following a hardware reset. Refer to the individual register definitions for details on reset (default) settings.

- The settings of the hardware context state variables are UNDEFINED. Software must program all state variables prior to their use in rendering.
- The ring buffer is disabled.
- All interrupts and error status bits are “masked” (disabled). All interrupts are disabled via IER. There will be no HW activity to cause any hardware/interrupt status bits to be set.
- The Hardware Status Page is located at 1FFFF000h (though HW status writes are effectively disabled)
- All FENCE registers are INVALID
- The GTT is disabled (accesses other than CPU reads, cursor and VGA reads will generate an error).
- All INSTDONE bits are set (“DONE”).
- The NOPID register is 0.
- All command groupings are enabled (via INSTPM)

13 Frame Buffer Compression ([DevCL] Only)

13.1 Overview

The Run-Length Encoded Frame Buffer Compression (RLE-FBC) function is a mechanism to reduce display refresh memory traffic. By reducing memory reads required for display refresh, power consumption is reduced (thus extending battery life for mobile systems).

The conditions under which the RLE-FBC is most effective are:

- Display images that are well suited to RLE compression. Good examples are text windows, slide shows, etc. Poor examples are 3D games - rich in textured and smooth-shaded objects.
- Screens that are fairly static. Good examples are screens with significant portions of the background showing, 2D apps (reading mail, etc.), CPU benchmarks, etc., or conditions when the CPU is idle. Poor examples are full-screen 3D games and benchmarks that flip the display image at or near display refresh rates.

Note that this compression function is different from, and mutually exclusive with, Discard Alpha Frame Buffer Compression – which is effective for 32bpp 3D environments.

The RLE-FBC function is comprised of three subfunctions:

- A **Compressor** that attempts to compress the display buffer as a background task.
- A **Decompressor** in the Display engine that uses compressed lines for display refresh, if available.
- A **Frame Buffer Write Detector** that snoops writes to the uncompressed frame buffer and invalidates the corresponding compressed lines.

The RLE-FBC **Compressor** periodically compresses lines of Display Plane A (an uncompressed display source image) using run-length encoding and stores the results into a pre-allocated compressed frame buffer. During subsequent display refreshes, the Display engine **Decompressor** attempts to refresh Display A from the compressed frame buffer. Lines that were not compressed or lines that were modified since the last compression – as detected by the **Frame Buffer Write Detector** – are displayed from the uncompressed buffer.

13.2 Programming Interface

13.2.1 FBC unit programming interface

The following table summarizes the register programming interface to the RLE-FBC function. Refer to the *Memory Interface Registers* chapter for details on the individual registers provided in the programming interface.

Register	Field(s)	Description
FBC_CFB_BASE	Compressed Frame Buffer Address	Specifies the location of the compressed frame buffer
FBC_LL_BASE	Compressed Line Length Buffer Address	Specifies the location of the compressed line length buffer
FBC_CONTROL	Enable	Turns the RLE-FBC function on/off
	Mode Select	Specifies Single or Periodic compression mode
	Interval	Specifies time period (in display refreshes) used in periodic mode
	Stop Compressing on Modification (DEBUG)	Specifies that the compression pass should be aborted if a line is modified during compression.
	Uncompressible Enable	Enable Uncompressible state for the tag RAM. if ENABLE compressor will mark the uncompressible scan line to prevent future compressing attempt
	Compressed Frame Buffer Stride	Specifies the stride (pitch) of the compressed frame buffer 64-byte unit
	Fence Number	Specifies the FENCE register associated with the uncompressed source frame buffer
FBC_CONTROL2		
	FBC Cx state mode	Specifies FBC behavior when PM signals CPU goes to Cx (non C0)
	CPU Fence Enable	If ENABLE the display buffer is existed within CPU fence
	Display Plane Select	Select Plane A or B for Frame Buffer Compression
FBC_YFENCE_DISP	Fence Display Buffer Y offset	Y offset from the CPU fence to the Display Buffer base
FBC_MOD_CTR	FBC modification Counter for Recompression	Recompress the Display Buffer only after the programmed number of modifications to the display buffer

Register	Field(s)	Description
FBC_COMMAND	Compression Request	Used to request compression passes in Single compression mode
FBC_STATUS	Compressing (RO)	Status indicating if the compressor is running.
	Compressed (RO, R/W for DEBUG)	Status indicating if the compressed frame buffer is available for display
	Any Modified (RO, R/W for DEBUG)	Indicates whether any lines of the uncompressed frame buffer have been modified since the last compression pass.
	Current Line Compressing (RO)	Indicates the progress of the compressor
FBC_TAG[0..N]	Tag[i+0..i+48] (DEBUG)	Status indication for each pair of display lines.

13.2.2 Programming interface from Display Engine

The following table summarizes the indirect register programming interface to the RLE-FBC function from Display Engine. These registers are programmed in Display Engine for Display function, but they are passed to FBC unit to use for Frame Buffer Compression operation. Depend on how FBC_CONTROL2< **Display Plane Select**> is set Display Plane A or B registers are passed to FBC unit. Refer to the *Memory Interface Registers* chapter for details on the individual registers provided in the programming interface.

FBC used these registers when reading uncompressed frame buffer and building a compressed buffer that is identical to uncompressed buffer of Display Plane A or B.

Register	Field(s)	Description
DSPA**(B)**CNTR	Display A**(B)** Source Pixel Format	4-bit source Pixel format- FBC can only works with 16-bit or 32-bit Source pixel format that organize in 8-bit chunk (not 10:10:10:2 format)
DSPA**(B)**STRIDE	Display A **(B)** Stride	This value is used to determine the line to line increment for the display. FBC can work with non-power-of-two stride from 2KB to 16KB with increment of 512bytes
DSPA**(B)**SURF	Display A **(B)** Surface Base Address	This address specifies the surface base address. When the surface is tiled, panning is specified using (x, y) offsets in the DSPA (B) TILEOFF register. This address must be 4K aligned.

Register	Field(s)	Description
DSPA**(B)**LINOFF	Plane Start Y-Position	These 12 bits specify the vertical position in lines of the beginning of the active display plane relative to the display surface.
	Plane Start X-Position	These 12 bits specify the horizontal offset in pixels of the beginning of the active display plane relative to the display surface.
HTOTAL**(B)**	Pipe A **(B)** Horizontal Active Display Pixels	This 12-bit field provides Horizontal Active Display resolutions up to 4096 pixels. Note that the first horizontal active display pixel is considered pixel number 0. The value programmed should be the (active pixels/line – 1).
VTOTAL**(B)**	Pipe A **(B)**Vertical Active Display Lines	This 12-bit field provides vertical active display resolutions up to 4096 lines. It should be programmed with the desired number of lines minus one.

13.3 Operating Modes

13.3.1 RLE-FBC Function Modes

The RLE-FBC function (compression and decompression) is enabled or disabled via the **Enable** bit of the FBC_CONTROL register.

In order to request the disabling of the function software must set **Enable** to DISABLED. The function does not subsequently become disabled until the **Compressing** status bit of FBC_STATUS is clear. Software must ensure that the function is in fact disabled (via interrogation of the **Compressing** status bit) before re-enabling the RLE-FBC function and under the following conditions:

- Prior to changing the contents of the FBC_CFB_BASE or FBC_LL_BASE registers
- Prior to changing the contents of the following fields of the FBC_CONTROL register:
  - Mode Select
  - Interval
  - Stop Compressing on Modification
  - Uncompressible Enable
  - Compressed Frame Buffer Stride
  - Fence Number
- Prior to changing the contents of the following fields of the FBC_CONTROL2 register:
  - FBC Cx state mode
  - CPU fence Enable
  - Frame Buffer Compression Display Plane Select A/B
- Prior to changing the contents of the FBC_Fence_Display_Y_Offset register:

- Prior to changing the contents of the following fields of the FBC_MOD_CTR register:
  - FBC_mod_ctr
  - FBC_mod_ctr_valid
- Prior to changing the display mode of the source frame buffer (Display Plane A) including display pixel format, dimensions, and pitch (stride).
- Prior to entering/use of any modes listed under *Restrictions* below

Modification of DEBUG-mode controls is implementation dependent.

13.3.2 Compression Modes

The RLE-FBC compressor is capable of operating in one of two modes, Single or Periodic Compression, as specified by the **Mode Select** field of the FBC_CONTROL register.

13.3.2.1 Single Compression Mode

In this mode software can request a single compression pass via the **Compression Request** bit of the FBC_COMMAND register. The compression results will be used until another compression is requested or the RLE-FBC function is disabled. Note that subsequent modifications to the uncompressed frame buffer will invalidate corresponding compressed lines – diminishing the benefits of the function.

Single compression mode is preferred when software has knowledge that significant portions of the frame buffer lines will remain static for a period of time – where memory bandwidth would not be wasted further recompressing the static frame buffer data.

13.3.2.2 Periodic Compression Mode

In Periodic mode, recompression is attempted at a programmed rate in units of display refreshes. The time period is programmed via the **Interval** field of the FBC_CONTROL register. The RLE-FBC compressor will not initiate a periodic compression if there have been no modifications to the source frame buffer since the last compression.

This mode is preferred when software expects significant portions of the frame buffer line to be written on a frequent basis (or at least cannot guarantee that this will not occur). The time period can be adjusted according to the refresh rate and/or frequency and extent of (expected) frame buffer modifications.

If Uncompressible **Enable** is set to ENABLED the compressor will mark a tag line uncompressible if both scan lines of a tag line are uncompressible so compressor won't attempt to compress these scan lines again in subsequent compression run unless these lines are modified by CPU or RC.

If FBC_mod_ctr_valid is SET the compressor will only attempt to recompress if the number of tag lines were modified since last compression run is greater or equal the value of FBC_mod_ctr.

13.4 Usage Restrictions

RLE Frame Buffer compression must not be enabled unless the following conditions are met:

1. If Display A is selected DSPACNTR—Display A Plane Control Register[Pixel Multiply] = No line duplication and Display A Plane Control Register[Horizontal Pixel Multiply] = 1x

2. If Display B is selected DSPBCNTR—Display B Plane Control Register[Pixel Multiply] = No line duplication and Display B Plane Control Register[Horizontal Pixel Multiply] = 1x

3. Panning of Selected Display Plane is permitted. If FBC is enabled and a compressed buffer is available when a panning event happened FBC will invalidate the current compressed buffer and recompress if necessary using the current FBC control parameters. If new uncompressed buffer required a new set of FBC control parameters then RLE-FBC must be first disabled.

4. Sync flips of Selected Display Plane are permitted. If FBC is enabled and a compressed buffer is available when sync flips event happened FBC will invalidate the current compressed buffer and recompress if necessary using the current FBC control parameters. If new uncompressed buffer required a new set of FBC control parameters then RLE-FBC must be first disabled

5. The display pixel format is 15-bit, 16-bit or 32-bit xRGB_8888 mode (as the alpha channel is removed as part of the compression).

6. Discard Alpha Frame Buffer Compression is DISABLED.

7. The uncompressed frame buffer is tiled with pitch from 2KB to 16KB in step of 0.5KB

8. The Line Width (in pixels) of the uncompressed frame buffer is a multiple of 8 in the range [640, 2048].

9. Number of lines of the uncompressed frame buffer is a multiple of 2 in the range [480, 1536].

10. Dual-wide display is not active.

11. If the pipe A is selected (i.e., DSPACNTR—Display A Plane Control Register [Display Pipe A Select] = Select Pipe A), then Pipe A Configuration Register [Interlaced modes] must be in Progressive mode.

12. If the pipe B is selected (i.e., DSPBCNTR—Display B Plane Control Register [Display Pipe B Select] = Select Pipe B), then Pipe B Configuration Register [Interlaced modes] must be in Progressive mode.

13. Compressed Frame Buffer Stride in bytes is equal or smaller than Uncompressed Frame Buffer Stride in bytes to prevent unintended buffer expansion in 16bpp frame.

14. Both Regular and SR display watermarks for 16bpp must equal 32bpp as calculated

15. Compressed Frame Buffer and Line Length buffers must reside entirely in stolen memory segment. If hardware tried to access compressed buffer or line length buffer outside of stolen memory FBC unit will be invalidate compressed buffers and makes unavailable to DISPLAY.

16. Display 180 degree rotation using gen4 hardware is turned off. This feature is not compatible with FBC scanline addressing. Software rotation can be enabled at the same time with FBC.

17. Async Flips are not permitted. FBC must be disabled when async flips are in use.

13.5 Power Management Interface

At the system level the amount of saving power of Frame Buffer Compression may be offset by power consumed by other units including CPU and memory subsystem when waiting for Frame Buffer Compression complete its pass. Device-specific power management modes need to add in to basic Frame Buffer Compression operation.

For [DevCL], different Cx state modes are used to provide a tuning mechanism between CPU low-power states (or Cx state) and FBC operation. Power Management Unit will signal to FBC that CPU is in low power state and wait for FBC to signal back that FBC is idle and no longer accessing external memory. Power Management unit then can implement global power saving scheme like putting external memory in self-refresh or clock gating FBC and/or other related units.

In DevCL, Cx state mode are specified as following:

- FBC_CONTROL2**<Cx state mode>**=IMMEDIATE IDLENESS. FBC blocks its requests to memory (read and write) and waits for all read returns to complete before asserting FBC-idle (default)
- FBC_CONTROL2**<Cx state mode>**=NORMAL IDLENESS. FBC finishes current compression pass before asserting FBC-idle
- FBC_CONTROL2**<Cx state mode>**=SCANLINE IDLENESS FBC completes the current line/line pair and skips remaining lines and makes the compressed buffer available for display before asserting FBC-idle.
- FBC_CONTROL2**<Cx state mode>**=IMMEDIATE DEBUG IDLENESS. FBC asserting FBC-idle immediately, more memory transactions may be still underway. This allows PM to find the fastest path to go to lower power state regardless of FBC operation.

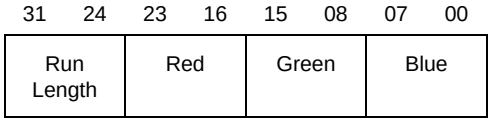
13.6 Memory Data Structures

13.6.1 RLE Pixel Runs

A compressed line contains one or more *pixel runs* of identical pixel values. A pixel run is stored as a DWord containing (1) an RGB *pixel value* and (2) a *run length* that specifies the number of times (minus one) that the pixel value is to be replicated.

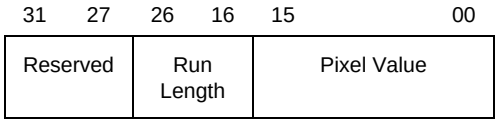
For 32bpp pixel formats, the run length is encoded in Bits 31:24 of the run Dword. This permits run lengths of 1 to 256 pixels. Any alpha value stored in Bits 31:24 is discarded. The remaining 24-bit RGB pixel value is left in place (in Bits 23:0).

Figure 13-1. 32bpp Pixel Run



For 16bpp pixel formats, the run length is encoded in Bits 26:16 of the run Dword. This permits run lengths of 1 to 2048 pixels. The 16-bit RGB pixel value is stored in Bits 15:0 (for 15bpp formats, Bit 15 is Reserved).

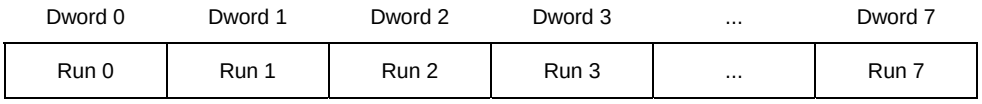
Figure 13-2. 16bpp Pixel Run



13.6.2 RLE Pixel Run Sets

The RLE-FBC function groups 8 consecutive pixel runs into 32-byte (Sword) *pixel run sets*. This matches the granularity used to read the compressed frame buffer.

Figure 13-3. Pixel Run Set



13.6.3 RLE-Compressed Line

An RLE-compressed *line* is comprised of a horizontal series of pixel run sets corresponding to a scan line in the uncompressed frame buffer.

Note that there is no encoding for “unused” Dwords in the last pixel run set. During display the Display engine will end the decompression of pixel runs when the number of decompressed pixels per line is satisfied.

13.6.4 RLE Compressed Frame and Line Length Buffers

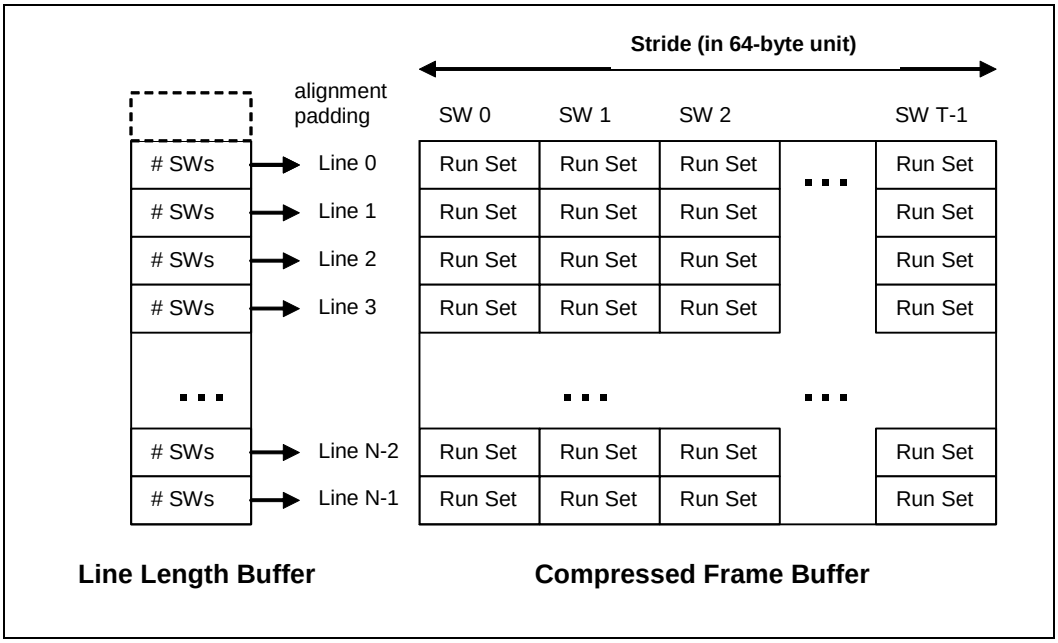
The RLE-compressed frame buffer and the Compressed Line Length Buffer must be in locked, fixed, contiguous, and uncacheable physical memory.

The RLE-Compressed Frame Buffer is a 4KB-aligned rectangular array of pixel run sets residing in physically contiguous memory (it is not mapped by the GTT). The physical address of the buffer is programmed via the FBC_CFB_BASE register.

The stride (width) of the buffer in Swords (run sets) is programmed via the **Compressed Frame Buffer Stride** field of the FBC_CONTROL register.

Different lines will typically compress to a different number of Pixel Runs. In order to record how many Swords needs to be fetched from the RLE-Compressed Frame Buffer, a Compressed Line Length Buffer is used. The Compressed Line Length Buffer is a (1536+32)-byte, 4KB-aligned list in physically contiguous memory (it is not mapped by the GTT). The physical address of the buffer is programmed via the FBC_LL_BASE register. Each byte in the buffer specifies the number of Swords (minus one) valid for the corresponding line in the RLE-Compressed Frame Buffer.

Figure 13-4. RLE-Compression Buffers



The byte in the Compressed Line Length Buffer that corresponds to Line 0 of the Compressed Frame Buffer is offset according to the alignment of the uncompressed display buffer. The Compressor and Decompressor both use the 6 least significant bit of y offset from Display Base as starting offset for line 0.

13.7 Tuning Parameters

13.7.1 Stride

The **Compressed Frame Buffer Stride** field of the FBC_CONTROL register specifies the distance (in 64-byte unit) between consecutive lines in the compressed frame buffer. If a source line cannot be compressed to fit within a compressed line, it will remain uncompressed.

The maximum compression ratio can be achieved by setting the compressed frame buffer stride to correspond with the uncompressed frame buffer line length. The stride can be set to a smaller number if there is not enough memory available for the compressed frame buffer.

13.7.2 Interval

As previously mentioned, the interval with which periodic compression passes are attempted can be adjusted as desired (e.g., as a function of refresh rate and/or expected frequency/extent of frame buffer modifications. The interval is programmed via the **Interval** field of the FBC_CONTROL register.

13.7.3 FBC Modification Counter

As previously mentioned, the FBC modification Counter can be used to reduce the number of recompression attempts if the number of modification since last attempt is small. At **Interval** expiry compressor will compare the number of accumulated tag line modifications (tag line modification counter) with the value of **FBC_mod_ctr** if the latter is larger the compressor will be back to sleep and tag line modification counter will continue counting.

13.8 Implementation (DEBUG)

This section describes the implementation of RLE_FBC function. Information in this section is not required for operational drivers – it is only required for debug activities.

13.8.1 Tag Array

A tag associated with every two sequential lines and indicates the current status of the lines. The tag states are defined as follows:

Tag Encoding	Definition	Description
'00'	Modified	At least one of the lines of the pair has been modified since the last compression pass, or a compression pass has not been made since (a) the source buffer address has changed, (b) RLE-FBC has been enabled, or (c) Reset
'01'	Uncompressed	One of the lines has not been compressed successfully.
'10'	Uncompressible	Both of the lines are uncompressible (compressed length is larger than compressed stride)
'11'	Compressed	Both of the lines are compressed

If the first line of the uncompressed source frame buffer is in an odd address, the first tag entry is associated with only one line, the first line; the second entry is associated with the second and third frame buffer line and so on. The last line will be also alone in this case.

13.8.1.1 Transitions

The following table describes the valid transitions of the Tag value. All tags start at the Modified state upon reset.

From	To	Conditions
Modified	Uncompressed	Unconditionally at the start of a compression pass.
Uncompressed	Modified	One of the lines is modified, or the source frame buffer base address was changed, or when compression becomes enabled.
Uncompressed	Compressed	Both lines were successfully compressed.
Uncompressed	Uncompressible	Both lines were unsuccessfully compressed in the previous pass
Compressed	Modified	Line was modified, or the source frame buffer base address was changed, or when compression becomes enabled.
Uncompressible	Modified	Line was modified, or the source frame buffer base address was changed, or when compression becomes enabled.

13.8.2 Compressor

The compressor will compress only if the display is on.

```
START:
if (Display Plane)
        return
on (Start of Display Vblank)
        Sample the FBC address and configuration registers
        if (Mode == Periodic)
                Interval counter = interval counter-- % Interval

        if ((Mode == Periodic AND Interval == 0) OR Compression Request)
AND
                Display is ON AND
                (At least one line pair is Modified) AND
                (!Compressing) AND
                (Local cache and write posting buffers are empty) AND
                (Display buffer is tiled)
                        goto COMPRESSION
                        else goto START

COMPRESSION:
        {
                Change Modified to Uncompressed // One cycle
                Set FBC_CONTROL<Compressing>
                Reset FBC_CONTROL<Compressed>
                Reset FBC_CONTROL<Modified>

                for (each and every Uncompressed line pair)
                {
          /* By first marking and then compressing we guarantee that
          modification to this line will be marked as Modified and will
          not be overridden when compression is completed */
                        Mark the pair as Compressed
                        Compress first line
                        if (Stride exceeded)
                                Mark pair as Uncompressed
                        else
                                Write the compressed line length to the
line-length buffer
                                Compress second line
                                if (Stride exceeded)
                                                Mark pair as
Uncompressed
                                else
                                                Write the
compressed line length to the line-length buffer
                                                Mark pair as
Compressed
                                  Set FBC_CONTROL<Compressed>
                } // end for each uncompressed line pair
                Reset the "Compression in progress" bit
                Set Compressed-buffer-avail bit
  } // end compression

// If we succeeded to compress or not
if (Mode == Periodic)
        Reset the interval-counter
goto START
```

13.8.3 Decompressor

When the display streamer gets the first line request it checks for the following condition:

- FBC_CONTROL<**Enable**> is set
- FBC_CONTROL<**Compressing**> is clear (compression not in progress)
- FBC_CONTROL<**Compressed**> is set (a compression pass has completed)

If any of these conditions are not met, only the uncompressed source buffer will be used for refresh.

If all these conditions are met, the Decompressor will, for every line:

- If the line marked as ***Compressed*** the display streamer will read the compressed line length from the compressed line length buffer, and then read the compressed line data according to this length. If the line is not marked as ***Compressed***, the display streamer reads the line from the uncompressed frame buffer. In both cases the pixel data is posted to the display FIFO.
- If the line is ***Compressed*** the Decompressor reads Dwords from the FIFO and sends on the pixel data multiple times according to the run length, 1 – 256 in 32-bit mode and 1 – 2048 in 16-bit modes. The Decompressor keeps track of the number of pixels and stops when it reaches the line width (in pixels) and discards any remaining Dwords.

13.8.4 Frame Buffer Write Detector

The Frame Buffer Write Detector snoops all frame buffers writes from the CPU and render engines, and marks the modified line pairs as ***Modified.***

- If Display buffer is a subset of the render buffer and cpu path is enabled via a fence, where the fence is a superset of the render buffer then frame buffers lines might be modified by both cpu write and render cache write.
- If Display buffer is a subset of render buffer and fence cpu path is disable then frame buffers might be modified by render cache line only.
- If CPU path is disabled and Render and Display are independent buffers then no modified should be happened.

In order to detect CPU write the following FBC registers need to be programmed before the FBC is enable

- FBC_CONTROL2 <**CPU Fence Enable**> is set.
- FBC_CONTROL <**Fence Number**> set to match the fence that render target and Frame Buffer reside in.
- FBC_YFENCE_DISP is set to the distance from fence base address to DSPA**(B)**SURF

Chipset unit passes CPU writes that are within Graphic Aperture to FBC. FBC write detector decode the line number and marked affected line as ***modified.***

There are no register programming needed for render cache write monitor. Render cache unit pass each write to its cache to FBC. If Render Target Address match with DSPA (B) SURF, and the render cache line has the same offset with active display then the affected line pair is marked as ***Modified.***

All lines will be marked as modified whenever:

- The uncompressed source Frame Buffer base address changes (this is only permitted to happen as a result of a direct register write – flips of Selected Display Plane are not allowed when RLE-FBC is enabled)
- RLE-FBC is enabled
- Reset

If the FBC_CONTROL<**Stop Compressing on Modification**> (DEBUG) bit is set, and a source frame buffer write is detected during a compression pass, the compression is aborted and the current line pair is marked as ***Modified***. Compression will be reattempted at the next periodic compression or when the next single compression pass is requested.

13.8.5 Coherency

The display coherency is kept by keeping the following rules:

- The compressed frame buffer is not displayed during compression.
- The Compressor will only compress lines that are marked as ***Uncompressed***.
- Lines state changes from ***Modified*** to ***Uncompressed*** can only when there are no display reads or pending display writes. This is achieved by waiting for Vblank start and then starting the compression only if the render cache is empty.
- Marking a line as ***Modified*** takes precedence over the (simultaneous) transition from ***Modified*** to ***Uncompressed***.
- Before a line pair is compressed, the tag is changed from ***Uncompressed*** to ***Compressed***. This will guarantee that if a line is modified while being compressed it will transition to the ***Modified*** state.
- Compressor frame buffer reads push CPU writes to memory.
- At the end of each compression path FBC issues dummy reads to push Compressed Buffer writes to memory.

14 BLT Engine

14.1 Introduction

2D Rendering can be divided into 2 categories: classical BLTs, described here, and 3D BLTs. 3D BLTs are operations which can take advantage of the 3D drawing engine's functionality and access patterns.

Functions such as Alpha BLTs, arithmetic (bilinear) stretch BLTs, rotations, transposing pixel maps, color space conversion, and DIBs are all considered 3D BLTs and are covered in the 3D rendering section. DIBs can be thought of as an indexed texture which uses the texture palette for performing the data translation. All drawing engines have swappable context. The same hardware can be used by multiple driver threads where the current state of the hardware is saved to memory and the appropriate state is loaded from memory on thread switches.

All operands for both 3D and classical BLTs can be in graphics aperture or cacheable system memory. Some operands can be immediates which are sent through the command stream. Immediate operands are: patterns, monochrome sources, DIB palettes, and DIB source operands. All non-monochrome operands which are not tiled have a stride granularity of a double-word (4 bytes).

The classical BLT commands support both linear addressing and X, Y coordinates with and without clipping. All X1 and Y1 destination and clipping coordinates are inclusive, while X2 and Y2 are exclusive. Currently, only destination coordinates can be negative. The source and clipping coordinates must be positive. If clipping is disabled, but a negative destination coordinate is specified, the negative coordinate is clipped to 0. Linear address BLT commands must supply a non-zero height and width. If either height or width = 0, then no accesses occur.

14.2 Classical BLT Engine Functional Description

The graphics controller provides a hardware-based BLT engine to off load the work of moving blocks of graphics data from the host CPU. Although the BLT engine is often used simply to copy a block of graphics data from the source to the destination, it also has the ability to perform more complex functions. The BLT engine is capable of receiving three different blocks of graphics data as input as shown in the figure below. The source data may exist in the frame buffer or the Graphics aperture. The pattern data always represents an 8x8 block of pixels that can be located in the frame buffer, Graphics aperture, or passed through a command packet. The pattern data must be located in linear memory.. The data already residing at the destination may also be used as an input. The destination data can also be located in the frame buffer or graphics aperture.

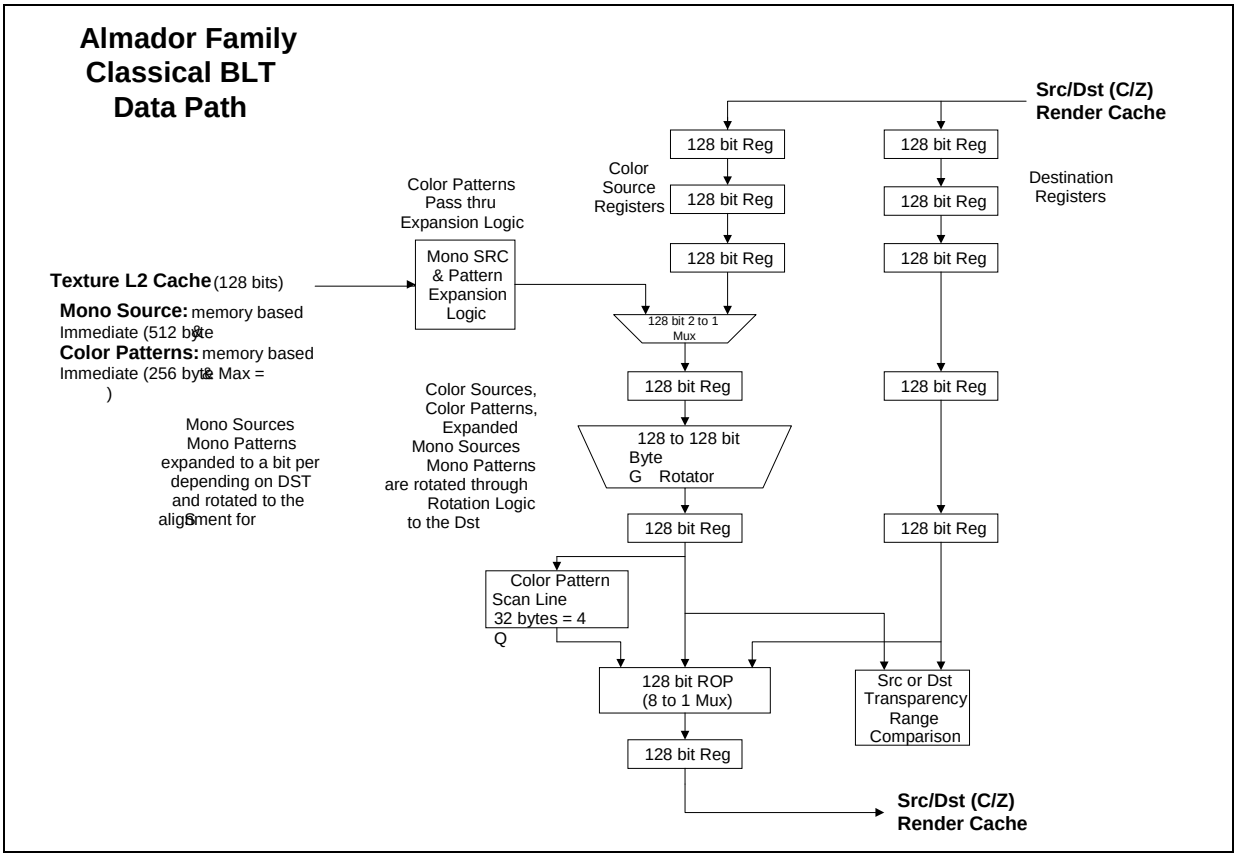


Figure 14-1. Block Diagram and Data Paths of the BLT Engine

The BLT engine may use any combination of these three different blocks of graphics data as operands, in both bit-wise logical operations to generate the actual data to be written to the destination, and in per-pixel write-masking to control the writing of data to the destination. It is intended that the BLT engine will perform these bit-wise and per-pixel operations on color graphics data that is at the same color depth that the rest of the graphics system has been set. However, if either the source or pattern data is monochrome, the BLT engine has the ability to put either block of graphics data through a process called “color expansion” that converts monochrome graphics data to color. Since the destination is often a location in the on-screen portion of the frame buffer, it is assumed that any data already at the destination will be of the appropriate color depth.

14.2.1 Basic BLT Functional Considerations

14.2.1.1 Color Depth Configuration and Color Expansion

The graphics system and BLT engine can be configured for color depths of 8, 16, and 32 bits per pixel.

The configuration of the BLT engine for a given color depth dictates the number of bytes of graphics data that the BLT engine will read and write for each pixel while performing a BLT operation. It is assumed that any graphics data already residing at

the destination which is used as an input is already at the color depth to which the BLT engine is configured. Similarly, it is assumed that any source or pattern data used as an input has this same color depth, unless one or both is monochrome. If either the source or pattern data is monochrome, the BLT engine performs a process called “color expansion” to convert such monochrome data to color at the color depth to which the BLT engine has been set.

During “color expansion” the individual bits of monochrome source or pattern data that correspond to individual pixels are converted into 1, 2, or 4 bytes (which ever is appropriate for the color depth to which the BLT engine has been set). If a given bit of monochrome source or pattern data carries a value of 1, then the byte(s) of color data resulting from the conversion process are set to carry the value of a specified foreground color. If a given bit of monochrome source or pattern data carries a value of 0, the resulting byte(s) are set to the value of a specified background color or not written if transparency is selected.

The BLT engine is set to a default configuration color depth of 8, 16, or 32 bits per pixel through BLT command packets. Whether the source and pattern data are color or monochrome must be specified using command packets. Foreground and background colors for the color expansion of both monochrome source and pattern data are also specified through the command packets. The source foreground and background colors used in the color expansion of monochrome source data are specified independently of those used for the color expansion of monochrome pattern data.

14.2.1.2 Graphics Data Size Limitations

The BLT engine is capable of transferring very large quantities of graphics data. Any graphics data read from and written to the destination is permitted to represent a number of pixels that occupies up to 65,536 scan lines and up to 32,768 bytes per scan line at the destination. The maximum number of pixels that may be represented per scan line’s worth of graphics data depends on the color depth.

Any source data used as an input must represent the same number of pixels as is represented by any data read from or written to the destination, and it must be organized so as to occupy the same number of scan lines and pixels per scan line.

The actual number of scan lines and bytes per scan line required to accommodate data read from or written to the destination are set in the destination width & height registers or using X and Y coordinates within the command packets. These two values are essential in the programming of the BLT engine, because the engine uses these two values to determine when a given BLT operation has been completed.

14.2.1.3 Bit-Wise Operations

The BLT engine can perform any one of 256 possible bit-wise operations using various combinations of the three previously described blocks of graphics data that the BLT engine can receive as input.

The choice of bit-wise operation selects which of the three inputs will be used, as well as the particular logical operation to be performed on corresponding bits from each of the selected inputs. The BLT engine automatically foregoes reading any form of graphics data that has not been specified as an input by the choice of bit-wise operation. An 8-bit code written to the raster operation field of the command packets

chooses the bit-wise operation. The following table lists the available bit-wise operations and their corresponding 8-bit codes.

Table 14-1. Bit-Wise Operations and 8-Bit Codes (00-3F)

Code	Value Written to Bits at Destination
00	writes all 0's
01	not(D or (P or S)))
02	D and (not(P or S))
03	not(P or S)
04	S and (not(D or P))
05	not(D or P)
06	not(P or (not(D xor S)))
07	not(P or (D and S))
08	S and (D and (notP))
09	not(P or (D xor S))
0A	D and (notP)
0B	not(P or (S and (notD)))
0C	S and (notP)
0D	not(P or (D and (notS)))
0E	not(P or (not(D or S)))
0F	notP
10	P and (not(D or S))
11	not(D or S)
12	not(S or (not(D xor P)))
13	not(S or (D and P))
14	not(D or (not(P xor S)))
15	not(D or (P and S))
16	P xor (S xor (D and (not(P and S))))
17	not(S xor ((S xor P) and (D xor S)))
18	(S xor P) and (P xor D)
19	not(S xor (D and (not(P and S))))
1A	P xor (D or (S and P))
1B	not(S xor (D and (P xor S)))
1C	P xor (S or (D and P))
1D	not(D xor (S and (P xor D)))
1E	P xor (D or S)
1F	not(P and (D or S))

Code	Value Written to Bits at Destination
20	D and (P and (notS))
21	not(S or(D xor P))
22	D and (notS)
23	not(S or (P and (notD)))
24	(S xor P) and (D xor S)
25	not(P xor (D and (not(S and P))))
26	S xor (D or (P and S))
27	S xor (D or (not(P xor S)))
28	D and (P xor S)
29	not(P xor (S xor (D or (P and S))))
2A	D and (not(P and S))
2B	not(S xor ((S xor P) and (P xor D)))
2C	S xor (P and (D or S))
2D	P xor (S or (notD))
2E	P xor (S or (D xor P))
2F	not(P and (S or (notD)))
30	P and (notS)
31	not(S or (D and (notP)))
32	S xor (D or (P or S))
33	notS
34	S xor (P or (D and S))
35	S xor (P or (not(D xor S)))
36	S xor (D or P)
37	not(S and (D or P))
38	P xor (S and (D or P))
39	S xor (P or (notD))
3A	S xor (P or (D xor S))
3B	not(S and (P or (notD)))
3C	P xor S
3D	S xor (P or (not(D or S)))
3E	S xor (P or (D and (notS)))
3F	not(P and S)

Notes: S = Source Data
P = Pattern Data
D = Data Already Existing at the Destination

Table 14-2. Bit-Wise Operations and 8-bit Codes (40 - 7F)

Code	Value Written to Bits at Destination
40	P and (S and (notD))
41	not(D or (P xor S))
42	(S xor D) and (P xor D)
43	not(S xor (P and (not(D and S))))
44	S and (notD)
45	not(D or (P and (notS)))
46	D xor (S or (P and D))
47	not(P xor (S and (D xor P)))
48	S and (D xor P)
49	not(P xor (D xor (S or (P and D))))
4A	D xor (P and (S or D))
4B	P xor (D or (notS))
4C	S and (not(D and P))
4D	not(S xor ((S xor P) or (D xor S)))
4E	P xor (D or (S xor P))
4F	not(P and (D or (notS)))
50	P and (notD)
51	not(D or (S and (notP)))
52	D xor (P or (S and D))
53	not(S xor (P and (D xor S)))
54	not(D or (not(P or S)))
55	notD
56	D xor (P or S)
57	not(D and (P or S))
58	P xor (D and (S or P))
59	D xor (P or (notS))
5A	D xor P
5B	D xor (P or (not(S or D)))
5C	D xor (P or (S xor D))
5D	not(D and (P or (notS)))
5E	D xor (P or (S and (notD)))
5F	not(D and P)

Code	Value Written to Bits at Destination
60	P and (D xor S)
61	not(D xor (S xor (P or (D and S))))
62	D xor (S and (P or D))
63	S xor (D or (notP))
64	S xor (D and (P or S))
65	D xor (S or (notP))
66	D xor S
67	S xor (D or (not(P or S)))
68	not(D xor (S xor (P or (not(D or S)))))
69	not(P xor (D xor S))
6A	D xor (P and S)
6B	not(P xor (S xor (D and (P or S))))
6C	S xor (D and P)
6D	not(P xor (D xor (S and (P or D))))
6E	S xor (D and (P or (notS)))
6F	not(P and (not(D xor S)))
70	P and (not(D and S))
71	not(S xor ((S xor D) and (P xor D)))
72	S xor (D or (P xor S))
73	not(S and (D or (notP)))
74	D xor (S or (P xor D))
75	not(D and (S or (notP)))
76	S xor (D or (P and (notS)))
77	not(D and S)
78	P xor (D and S)
79	not(D xor (S xor (P and (D or S))))
7A	D xor (P and (S or (notD)))
7B	not(S and (not(D xor P)))
7C	S xor (P and (D or (notS)))
7D	not(D and (not(P xor S)))
7E	(S xor P) or (D xor S)
7F	not(D and (P and S))

Notes: S = Source Data
P = Pattern Data
D = Data Already Existing at the Destination

Table 14-3. Bit-Wise Operations and 8-bit Codes (80 - BF)

Code	Value Written to Bits at Destination
80	D and (P and S)
81	not((S xor P) or (D xor S))
82	D and (not(P xor S))
83	not(S xor (P and (D or (notS))))
84	S and (not(D xor P))
85	not(P xor (D and (S or (notP))))
86	D xor (S xor (P and (D or S)))
87	not(P xor (D and S))
88	D and S
89	not(S xor (D or (P and (notS))))
8A	D and (S or (notP))
8B	not(D xor (S or (P xor D)))
8C	S and (D or (notP))
8D	not(S xor (D or (P xor S)))
8E	S xor ((S xor D) and (P xor D))
8F	not(P and (not(D and S)))
90	P and (not(D xor S))
91	not(S xor (D and (P or (notS))))
92	D xor (P xor (S and (D or P)))
93	not(S xor (P and D))
94	P xor (S xor (D and (P or S)))
95	not(D xor (P and S))
96	D xor (P xor S)
97	P xor (S xor (D or (not(P or S))))
98	not(S xor (D or (not(P or S))))
99	not(D xor S)
9A	D xor (P and (notS))
9B	not(S xor (D and (P or S)))
9C	S xor (P and (notD))
9D	not(D xor (S and (P or D)))
9E	D xor (S xor (P or (D and S)))
9F	not(P and (D xor S))

Code	Value Written to Bits at Destination
A0	D and P
A1	not(P xor (D or (S and (notP))))
A2	D and (P or (notS))
A3	not(D xor (P or (S xor D)))
A4	not(P xor (D or (not(S or P))))
A5	not(P xor D)
A6	D xor (S and (notP))
A7	not(P xor (D and (S or P)))
A8	D and (P or S)
A9	not(D xor (P or S))
AA	D
AB	D or (not(P or S))
AC	S xor (P and (D xor S))
AD	not(D xor (P or (S and D)))
AE	D or (S and (notP))
AF	D or (notP)
B0	P and (D or (notS))
B1	not(P xor (D or (S xor P)))
B2	S xor ((S xor P) or (D xor S))
B3	not(S and (not(D and P)))
B4	P xor (S and (notD))
B5	not(D xor (P and (S or D)))
B6	D xor (P xor (S or (D and P)))
B7	not(S and (D xor P))
B8	P xor (S and (D xor P))
B9	not(D xor (S or (P and D)))
BA	D or (P and (notS))
BB	D or (notS)
BC	S xor (P and (not(D and S)))
BD	not((S xor D) and (P xor D))
BE	D or (P xor S)
BF	D or (not(P and S))

Notes: S = Source Data
P = Pattern Data
D = Data Already Existing at the Destination

Table 14-4. Bit-Wise Operations and 8-bit Codes (C0 - FF)

Code	Value Written to Bits at Destination
C0	P and S
C1	not(S xor (P or (D and (notS))))
C2	not(S xor (P or (not(D or S))))
C3	not(P xor S)
C4	S and (P or (notD))
C5	not(S xor (P or (D xor S)))
C6	S xor (D and (notP))
C7	not(P xor (S and (D or P)))
C8	S and (D or P)
C9	not(S xor (P or D))
CA	D xor (P and (S xor D))
CB	not(S xor (P or (D and S)))
CC	S
CD	S or (not(D or P))
CE	S or (D and (notP))
CF	S or (notP)
D0	P and (S or (notD))
D1	not(P xor (S or (D xor P)))
D2	P xor (D and (notS))
D3	not(S xor (P and (D or S)))
D4	S xor ((S xor P) and (P xor D))
D5	not(D and (not(P and S)))
D6	P xor (S xor (D or (P and S)))
D7	not(D and (P xor S))
D8	P xor (D and (S xor P))
D9	not(S xor (D or (P and S)))
DA	D xor (P and (not(S and D)))
DB	not((S xor P) and (D xor S))
DC	S or (P and (notD))
DD	S or (notD)
DE	S or (D xor P)
DF	S or (not(D and P))

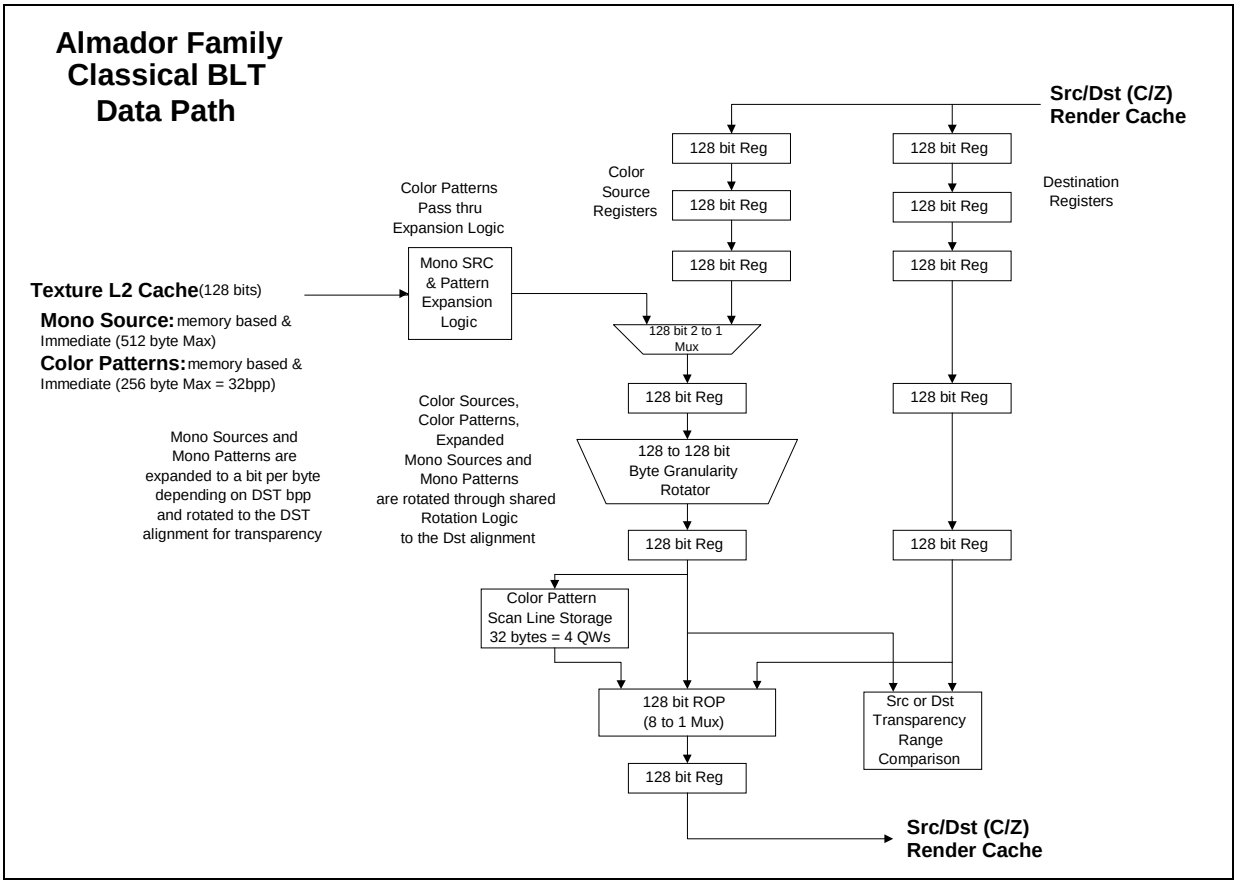
Code	Value Written to Bits at Destination
E0	P and (D or S)
E1	not(P xor (D or S))
E2	D xor (S and (P xor D))
E3	not(P xor (S or (D and P)))
E4	S xor (D and (P xor S))
E5	not(P xor (D or (S and P)))
E6	S xor (D and (not(P and S)))
E7	not((S xor P) and (P xor D))
E8	S xor ((S xor P) and (D xor S))
E9	not(D xor (S xor (P and (not(D and S)))))
EA	D or (P and S)
EB	D or (not(P xor S))
EC	S or (D and P)
ED	S or (not(D xor P))
EE	D or S
EF	S or (D or (notP))
F0	P
F1	P or (not(D or S))
F2	P or (D and (notS))
F3	P or (notS)
F4	P or (S and (notD))
F5	P or (notD)
F6	P or (D xor S)
F7	P or (not(D and S))
F8	P or (D and S)
F9	P or (not(D xor S))
FA	D or P
FB	D or (P or (notS))
FC	P or S
FD	P or (S or (notD))
FE	D or (P or S)
FF	writes all 1's

Notes: S = Source Data
P = Pattern Data
D = Data Already Existing at the Destination

14.2.1.4 Per-Pixel Write-Masking Operations

The BLT engine is able to perform per-pixel write-masking with various data sources used as pixel masks to constrain which pixels at the destination are to be written to by the BLT engine. As shown in the figure below, either monochrome source or monochrome pattern data may be used as pixel masks. Color pattern data cannot be used. Another available pixel mask is derived by comparing a particular color range per color channel to either the color already specified for a given pixel at the destination or source.

Figure 14-2. Block Diagram and Data Paths of the BLT Engine



The command packets can specify the monochrome source or the monochrome pattern data as a pixel mask. When this feature is used, the bits that carry a value of 0 cause the bytes of the corresponding pixel at the destination to not be written to by the BLT engine, thereby preserving whatever data was originally carried within those bytes. This feature can be used in writing characters to the display, while also preserving the pre-existing backgrounds behind those characters. When both operands are in the transparent mode, the logical AND of the 2 operands are used for the write enables per pixel.

The 3-bit field, destination transparency mode, within the command packets can select per-pixel write-masking with a mask based on the results of color comparisons. The monochrome source background and foreground are range compared with either the bytes for the pixels at the destination or the source operand. This operation is described in the BLT command packet and register descriptions.

14.2.1.5 When the Source and Destination Locations Overlap

It is possible to have BLT operations in which the locations of the source and destination data overlap. This frequently occurs in BLT operations where a user is shifting the position of a graphical item on the display by only a few pixels. In these situations, the BLT engine must be programmed so that destination data is not written into destination locations that overlap with source locations before the source data at those locations has been read. Otherwise, the source data will become corrupted. The XY commands determine whether there is an overlap and perform the accesses in the proper direction to avoid data corruption.

The following figure shows how the source data can be corrupted when a rectangular block is copied from a source location to an overlapping destination location. The BLT engine typically reads from the source location and writes to the destination location starting with the left-most pixel in the top-most line of both, as shown in step (a). As shown in step (b), corruption of the source data has already started with the copying of the top-most line in step (a) — part of the source that originally contained lighter-colored pixels has now been overwritten with darker-colored pixels. More source data corruption occurs as steps (b) through (d) are performed. At step (e), another line of the source data is read, but the two right-most pixels of this line are in the region where the source and destination locations overlap, and where the source has already been overwritten as a result of the copying of the top-most line in step (a). Starting in step (f), darker-colored pixels can be seen in the destination where lighter-colored pixels should be. This errant effect occurs repeatedly throughout the remaining steps in this BLT operation. As more lines are copied from the source location to the destination location, it becomes clear that the end result is not what was originally intended.

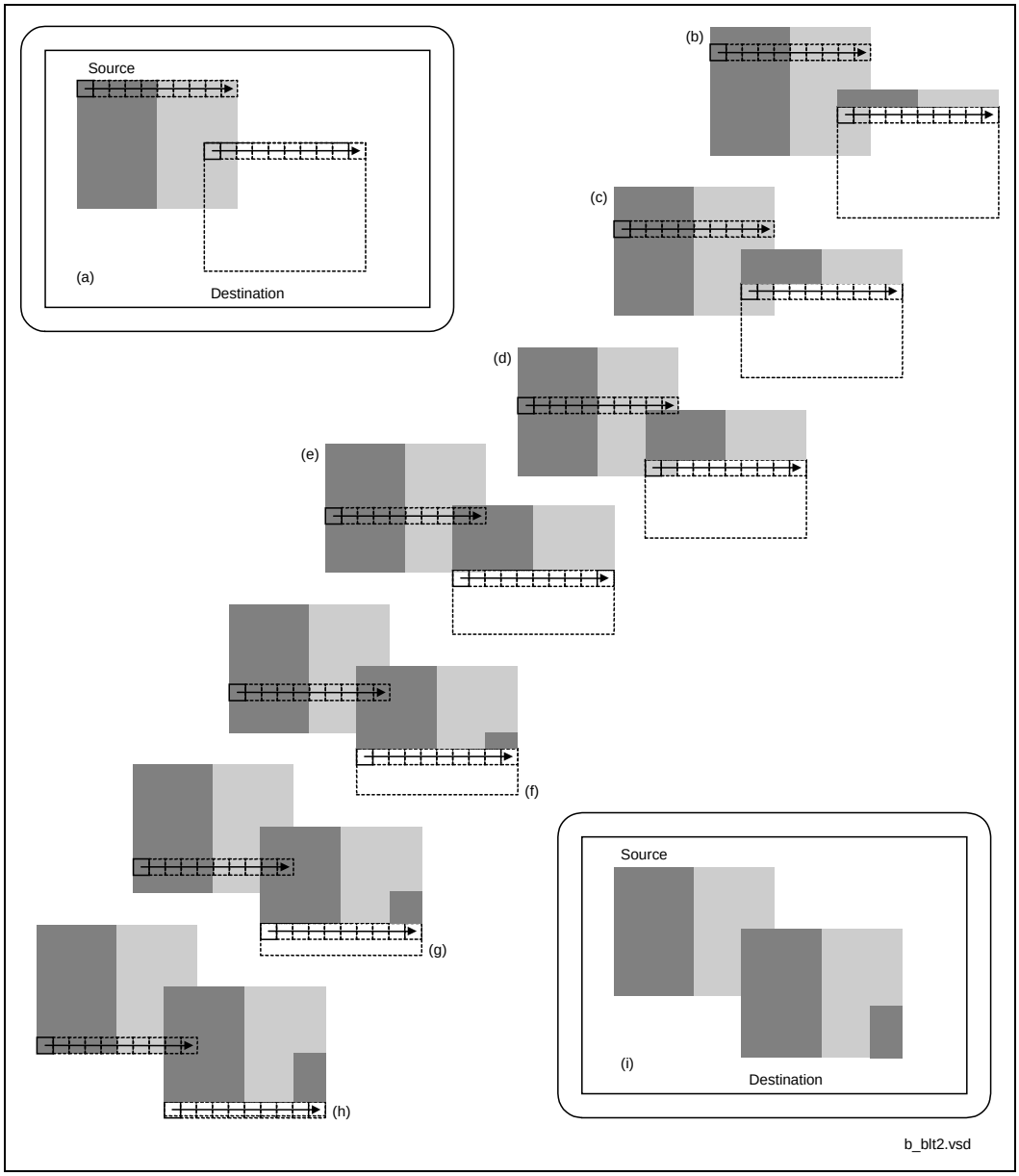


Figure 14-3. Source Corruption in BLT with Overlapping Source and Destination Locations

The BLT engine can alter the order in which source data is read and destination data is written when necessary to avoid source data corruption problems when the source and destination locations overlap. The command packets provide the ability to change the point at which the BLT engine begins reading and writing data from the upper left-hand corner (the usual starting point) to one of the other three corners. The BLT engine may be set to read data from the source and write it to the destination starting at any of the four corners of the panel.

The XY command packets perform the necessary comparisons and start at the proper corner of each operand which avoids data corruption.

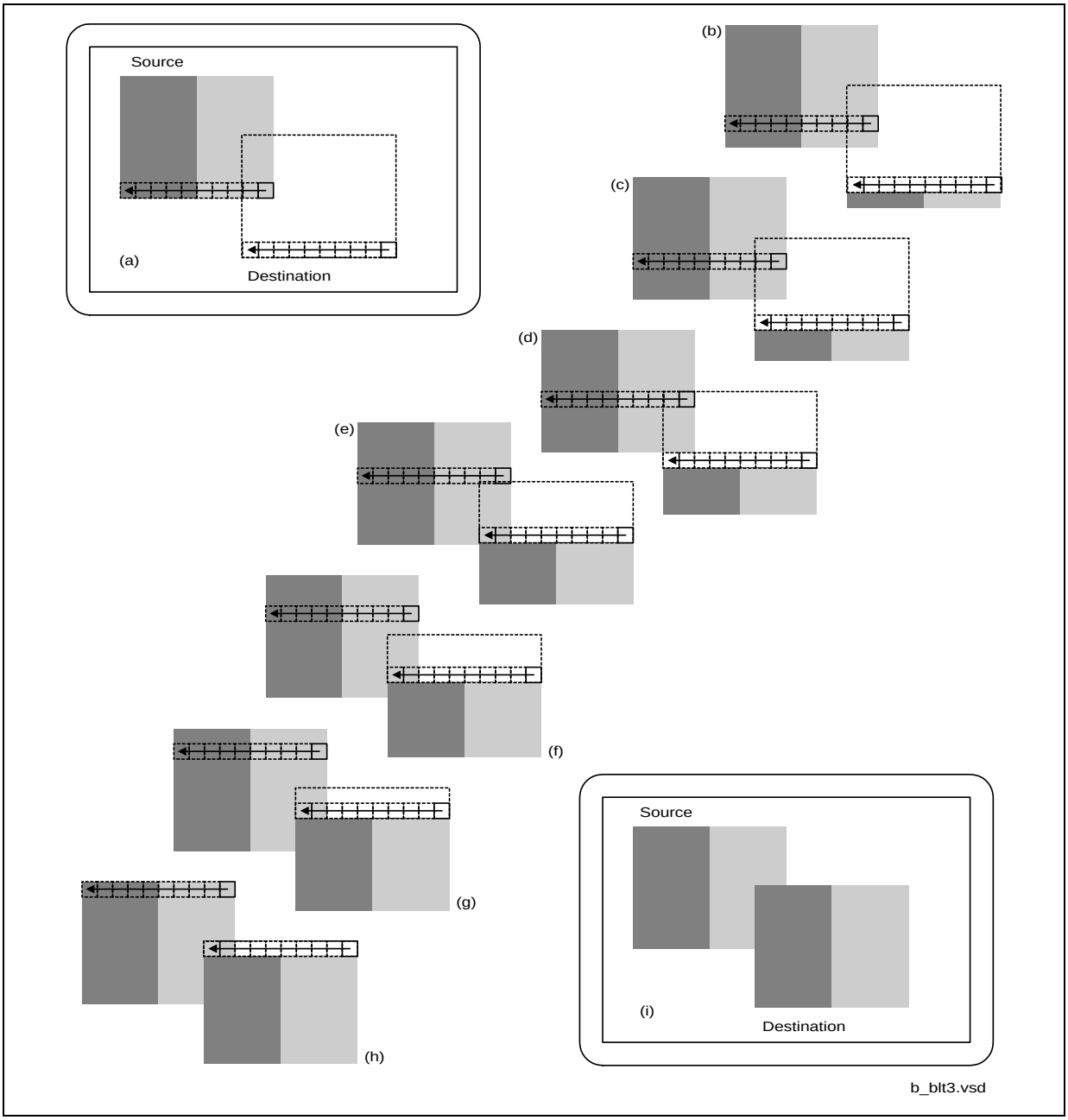
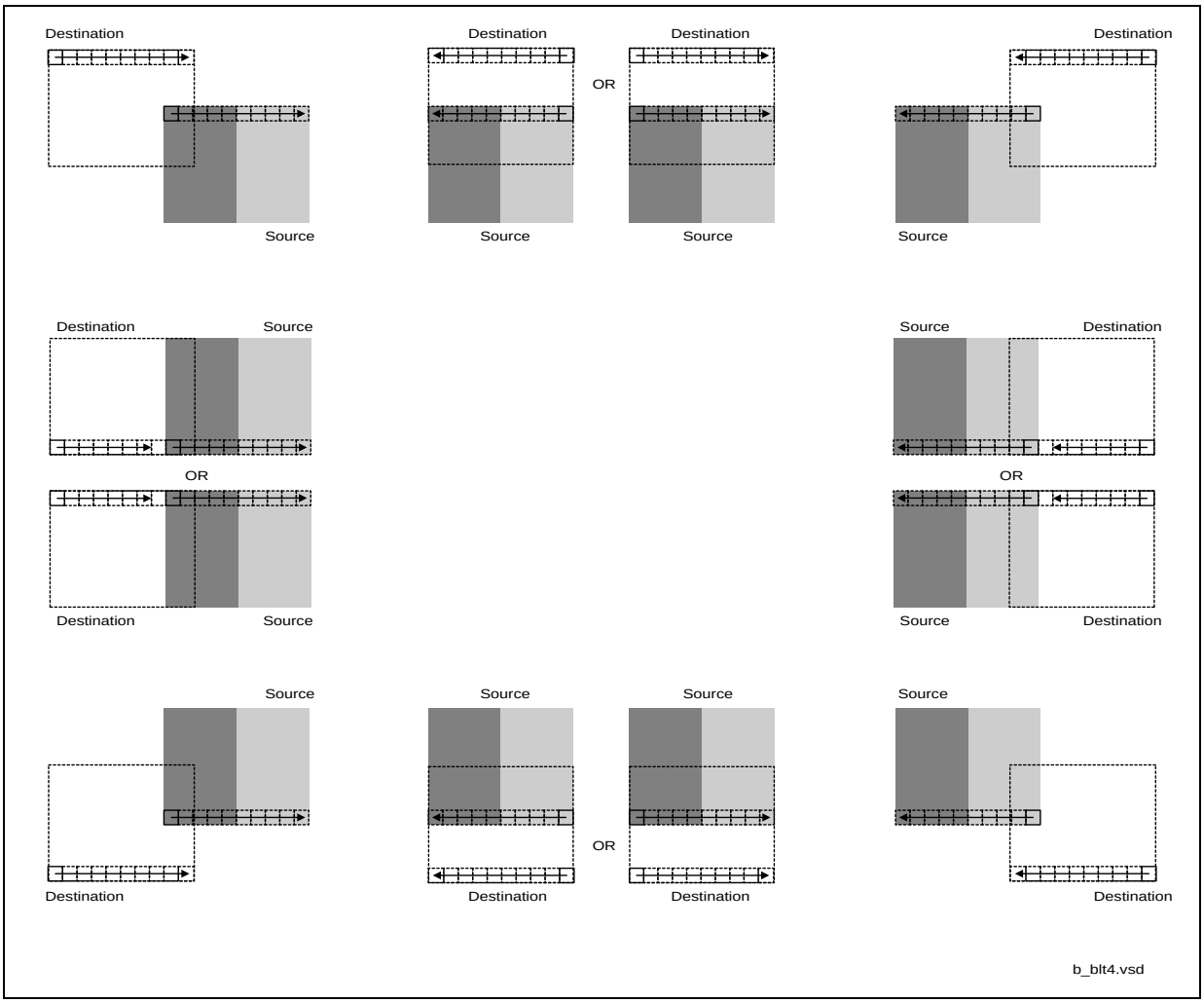


Figure 14-4. Correctly Performed BLT with Overlapping Source and Destination Locations

The following figure illustrates how this feature of the BLT engine can be used to perform the same BLT operation as was illustrated in the figure above, while avoiding the corruption of source data. As shown in the figure below, the BLT engine reads the source data and writes the data to the destination starting with the right-most pixel of the bottom-most line. By doing this, no pixel existing where the source and destination locations overlap will ever be written to before it is read from by the BLT engine. By the time the BLT operation has reached step (e) where two pixels existing where the source and destination locations overlap are about to be over written, the source data for those two pixels has already been read.

Figure 14-5. Suggested Starting Points for Possible Source and Destination Overlap Situations



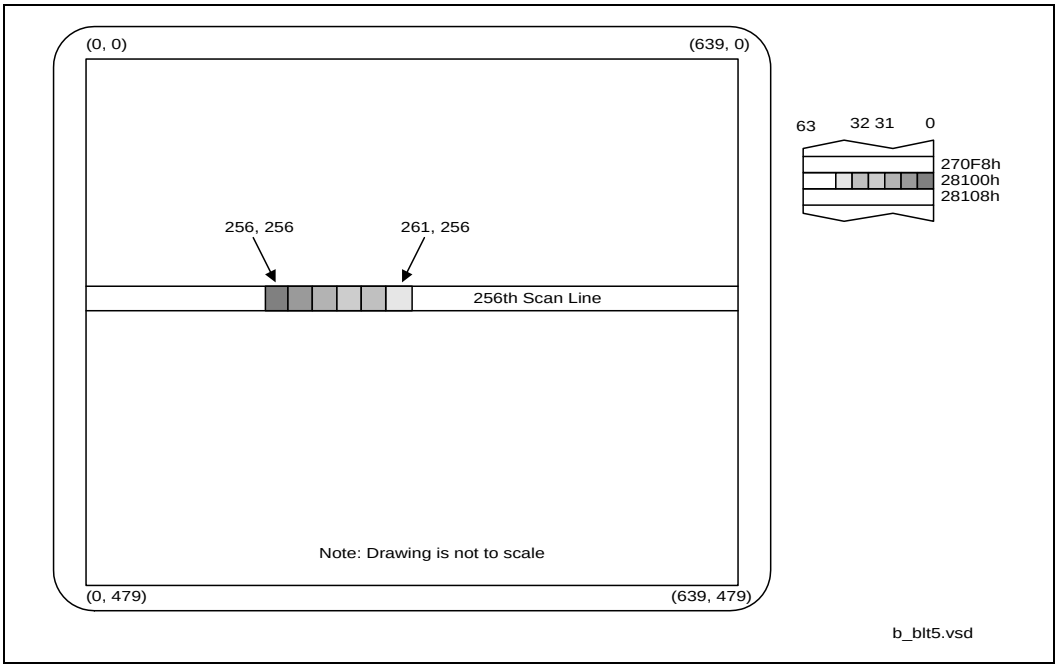
The figure above shows the recommended lines and pixels to be used as starting points in each of 8 possible ways in which the source and destination locations may overlap. In general, the starting point should be within the area in which the source and destination overlap.

14.2.2 Basic Graphics Data Considerations

14.2.2.1 Contiguous vs. Discontinuous Graphics Data

Graphics data stored in memory, particularly in the frame buffer of a graphics system, has organizational characteristics that often distinguish it from other varieties of data. The main distinctive feature is the tendency for graphics data to be organized in a discontinuous block of graphics data made up of multiple sub-blocks of bytes, instead of a single contiguous block of bytes.

Figure 14-6. Representation of On-Screen Single 6-Pixel Line in the Frame Buffer



The figure above shows an example of contiguous graphics data — a horizontal line made up of six adjacent pixels within a single scan line on a display with a resolution of 640x480. Presuming that the graphics system driving this display has been set to 8 bits per pixel and that the frame buffer's starting address of 0h corresponds to the upper left-most pixel of this display, then the six pixels that make this horizontal line starting at coordinates (256, 256) occupies the six bytes starting at frame buffer address 28100h, and ending at address 28105h.

In this case, there is only one scan line's worth of graphics data in this single horizontal line, so the block of graphics data for all six of these pixels exists as a single, contiguous block comprised of only these six bytes. The starting address and the number of bytes are the only pieces of information that a BLT engine would require to read this block of data.

The simplicity of the above example of a single horizontal line contrasts sharply to the example of discontinuous graphics data depicted in the figure below. The simple six-pixel line of the figure above is now accompanied by three more six-pixel lines placed on subsequent scan lines, resulting in the 6x4 block of pixels shown.

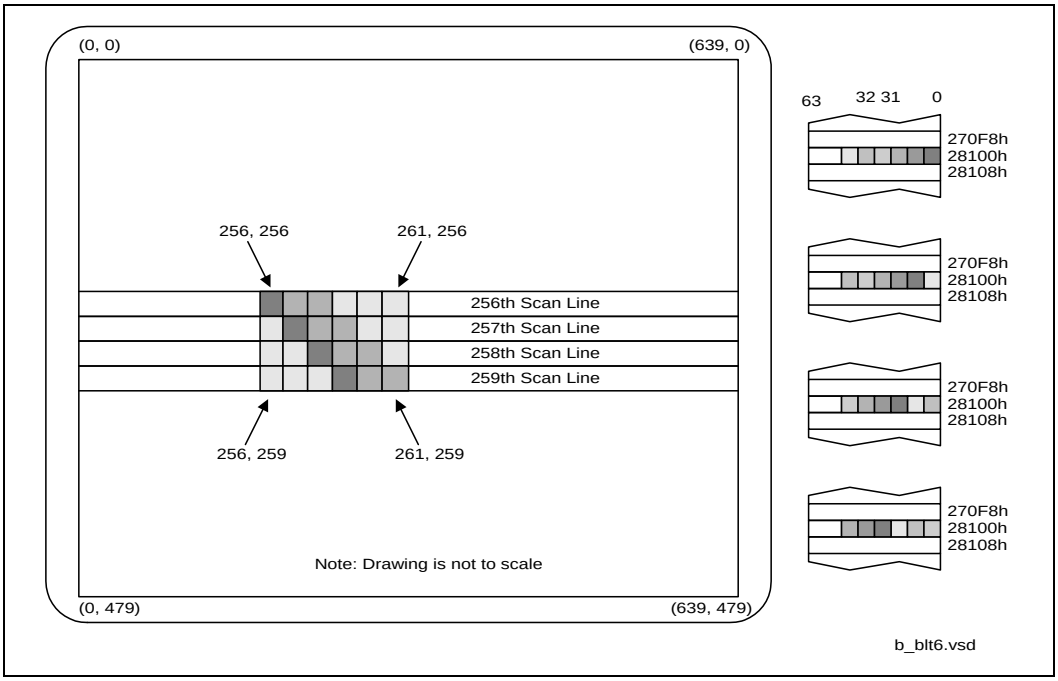


Figure 14-7. Representation of On-Screen 6x4 Array of Pixels in the Frame Buffer

Since there are other pixels on each of the scan lines on which this 6x4 block exists that are not part of this 6x4 block, what appears to be a single 6x4 block of pixels on the display must be represented by a discontinuous block of graphics data made up of 4 separate sub-blocks of six bytes apiece in the frame buffer at addresses 28100h, 28380h, 28600h, and 28880h. This situation makes the task of reading what appears to be a simple 6x4 block of pixels more complex. However, there are two characteristics of this 6x4 block of pixels that help simplify the task of specifying the locations of all 24 bytes of this discontinuous block of graphics data: all four of the sub-blocks are of the same length, and the four sub-blocks are separated from each other at equal intervals.

The BLT engine is designed to make use of these characteristics of graphics data to simplify the programming required to handle discontinuous blocks of graphics data. For such a situation, the BLT engine requires only four pieces of information: the starting address of the first sub-block, the length of a sub-block, the offset (in bytes), pitch, of the starting address of each subsequent sub-block, and the quantity of sub-blocks.

14.2.2.2 Source Data

The source data may exist in the frame buffer or elsewhere in the graphics aperture where the BLT engine may read it directly, or it may be provided to the BLT engine by the host CPU through the command packets. The block of source graphics data may be either contiguous or discontinuous, and may be either in color (with a color depth that matches that to which the BLT engine has been set) or monochrome.

The source select bit in the command packets specifies whether the source data exists in the frame buffer or is provided through the command packets. Monochrome source data is always specified as being supplied through an immediate command packet.

If the color source data resides within the frame buffer or elsewhere in the graphics aperture, then the Source Address Register, specified in the command packets is used to specify the address of the source.

In cases where the host CPU provides the source data, it does so by writing the source data to ring buffer directly after the BLT command that requires the data or uses an IMMEDIATE_INDIRECT_BLT command packet which has a size and pointer to the operand in Graphics aperture.

The block of bytes sent by the host CPU through the command packets must be quadword-aligned and the source data contained within the block of bytes must also be aligned.

To accommodate discontinuous source data, the source and destination pitch registers can be used to specify the offset in bytes from the beginning of one scan line's worth source data to the next. Otherwise, if the source data is contiguous, then an offset equal to the length of a scan line's worth of source data should be specified.

14.2.2.3 Monochrome Source Data

The opcode of the command packet specifies whether the source data is color or monochrome. Since monochrome graphics data only uses one bit per pixel, each byte of monochrome source data typically carries data for 8 pixels which hinders the use of byte-oriented parameters when specifying the location and size of valid source data. Some additional parameters must be specified to ensure the proper reading and use of monochrome source data by the BLT engine. The BLT engine also provides additional options for the manipulation of monochrome source data versus color source data.

The various bit-wise logical operations and per-pixel write-masking operations were designed to work with color data. In order to use monochrome data, the BLT engine converts it into color through a process called color expansion, which takes place as a BLT operation is performed. In color expansion the single bits of monochrome source data are converted into one, two, or four bytes (depending on the color depth) of color data that are set to carry value corresponding to either the foreground or background color that have been specified for use in this conversion process. If a given bit of monochrome source data carries a value of 1, then the byte(s) of color data resulting from the conversion process will be set to carry the value of the foreground color. If a given bit of monochrome source data carries a value of 0, then the resulting byte(s) will be set to the value of the background color. The foreground and background colors used in the color expansion of monochrome source data can be set in the source expansion foreground color register and the source expansion background color register.

The BLT Engine requires that the bit alignment of each scan line's worth of monochrome source data be specified. Each scan line's worth of monochrome source data is word aligned but can actually start on any bit boundary of the first byte. Monochrome text is special cased and it is bit or byte packed, where in bit packed there are no invalid pixels (bits) between scan lines. There is a 3 bit field which indicates the starting pixel position within the first byte for each scan line, Mono Source Start.

The BLT engine also provides various clipping options for use with specific BLT commands (BLT_TEXT) with a monochrome source. Clipping is supported through: Clip rectangle Y addresses or coordinates and X coordinates along with scan line

starting and ending addresses (with Y addresses) along with X starting and ending coordinates.

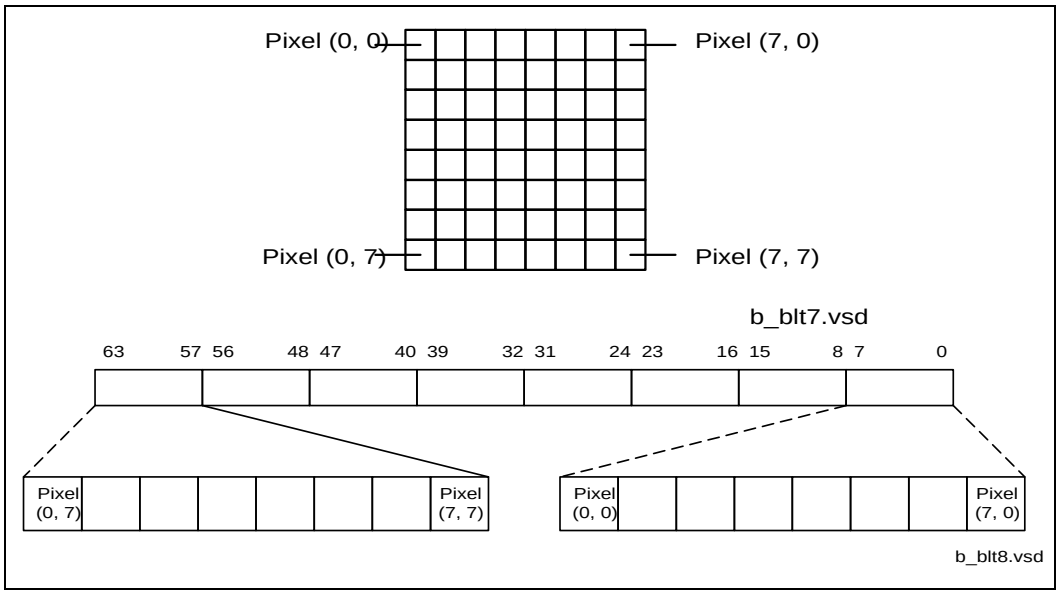
The maximum immediate source size is 128 bytes.

14.2.2.4 Pattern Data

The color pattern data must exist within the frame buffer or Graphics aperture where the BLT engine may read it directly or it can be sent through the command stream. The pattern data must be located in linear memory. Monochrome pattern data is supplied by the command packet when it is to be used. As shown in figure below, the block of pattern graphics data always represents a block of 8x8 pixels. The bits or bytes of a block of pattern data may be organized in the frame buffer memory in only one of three ways, depending upon its color depth which may be 8, 16, or 32 bits per pixel (whichever matches the color depth to which the BLT engine has been set), or monochrome.

The maximum color pattern size is 256 bytes.

Figure 14-8. Pattern Data -- Always an 8x8 Array of Pixels



The Pattern Address Register is used to specify the address of the color pattern data at which the block of pattern data begins. The three least significant bits of the address written to this register are ignored, because the address must be in terms of quadwords. This is because the pattern must always be located on an address boundary equal to its size. Monochrome patterns take up 8 bytes, or a single quadword of space, and are loaded through the command packet that uses it. Similarly, color patterns with color depths of 8, 16, and 32 bits per pixel must start on 64-byte, 128-byte and 256-byte boundaries, respectively. The next 3 figures show how monochrome, 8bpp, 16bpp, and 32bpp pattern data , respectively, is organized in memory.

Figure 14-9. 8bpp Pattern Data -- Occupies 64 Bytes (8 quadwords)

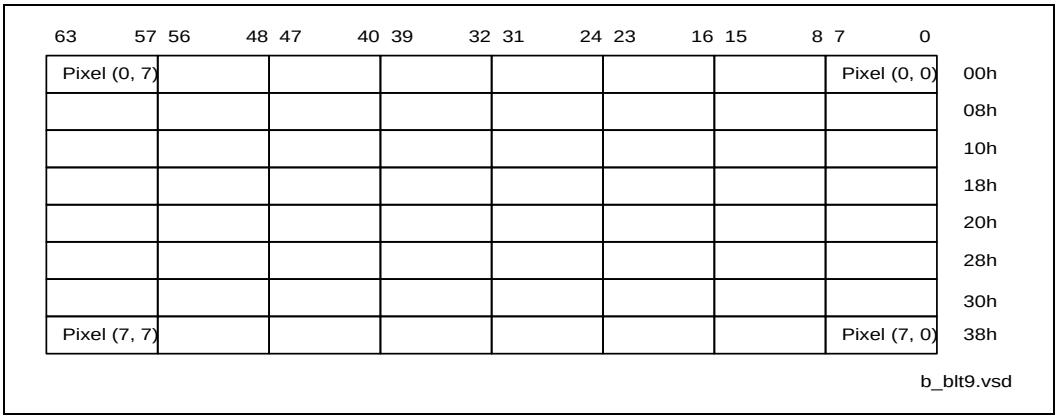


Figure 14-10. 16bpp Pattern Data -- Occupies 128 Bytes (16 quadwords)

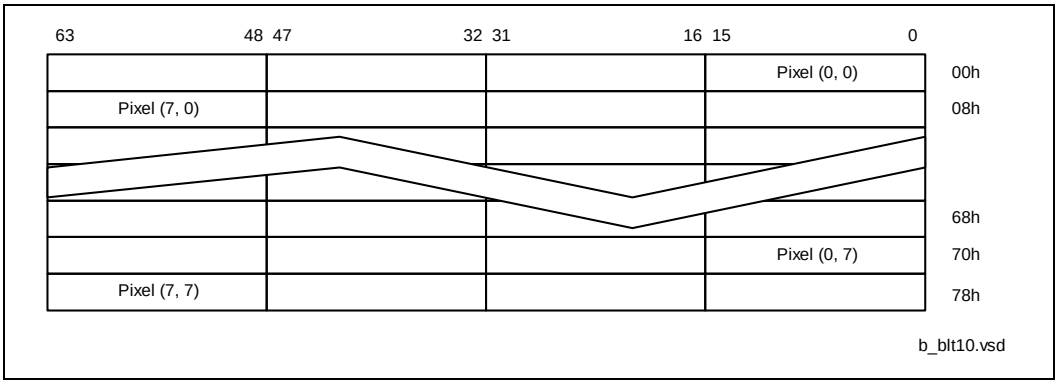
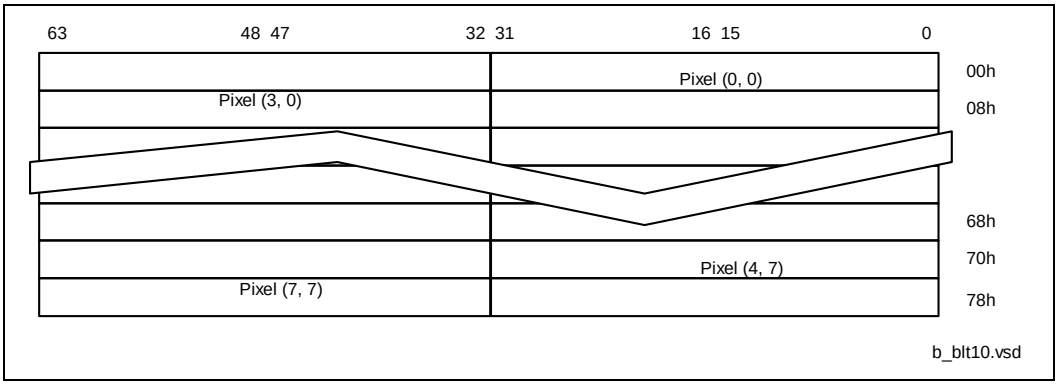


Figure 14-11. 32bpp Pattern Data -- Occupies 256 Bytes (32 quadwords)



The opcode of the command packet specifies whether the pattern data is color or monochrome. The various bit-wise logical operations and per-pixel write-masking operations were designed to work with color data. In order to use monochrome pattern data, the BLT engine is designed to convert it into color through a process called “color expansion” which takes place as a BLT operation is performed. In color expansion, the single bits of monochrome pattern data are converted into one, two, or four bytes (depending on the color depth) of color data that are set to carry values

corresponding to either the foreground or background color that have been specified for use in this process. The foreground color is used for pixels corresponding to a bit of monochrome pattern data that carry the value of 1, while the background color is used where the corresponding bit of monochrome pattern data carries the value of 0. The foreground and background colors used in the color expansion of monochrome pattern data can be set in the Pattern Expansion Foreground Color Register and Pattern Expansion Background Color Register.

14.2.2.5 Destination Data

There are actually two different types of “destination data”: the graphics data already residing at the location that is designated as the destination, and the data that is to be written into that very same location as a result of a BLT operation.

The location designated as the destination must be within the frame buffer or Graphics aperture where the BLT engine can read from it and write to it directly. The blocks of destination data to be read from and written to the destination may be either contiguous or discontinuous. All data written to the destination will have the color depth to which the BLT engine has been set. It is presumed that any data already existing at the destination which will be read by the BLT engine will also be of this same color depth — the BLT engine neither reads nor writes monochrome destination data.

The Destination Address Register is used to specify the address of the destination.

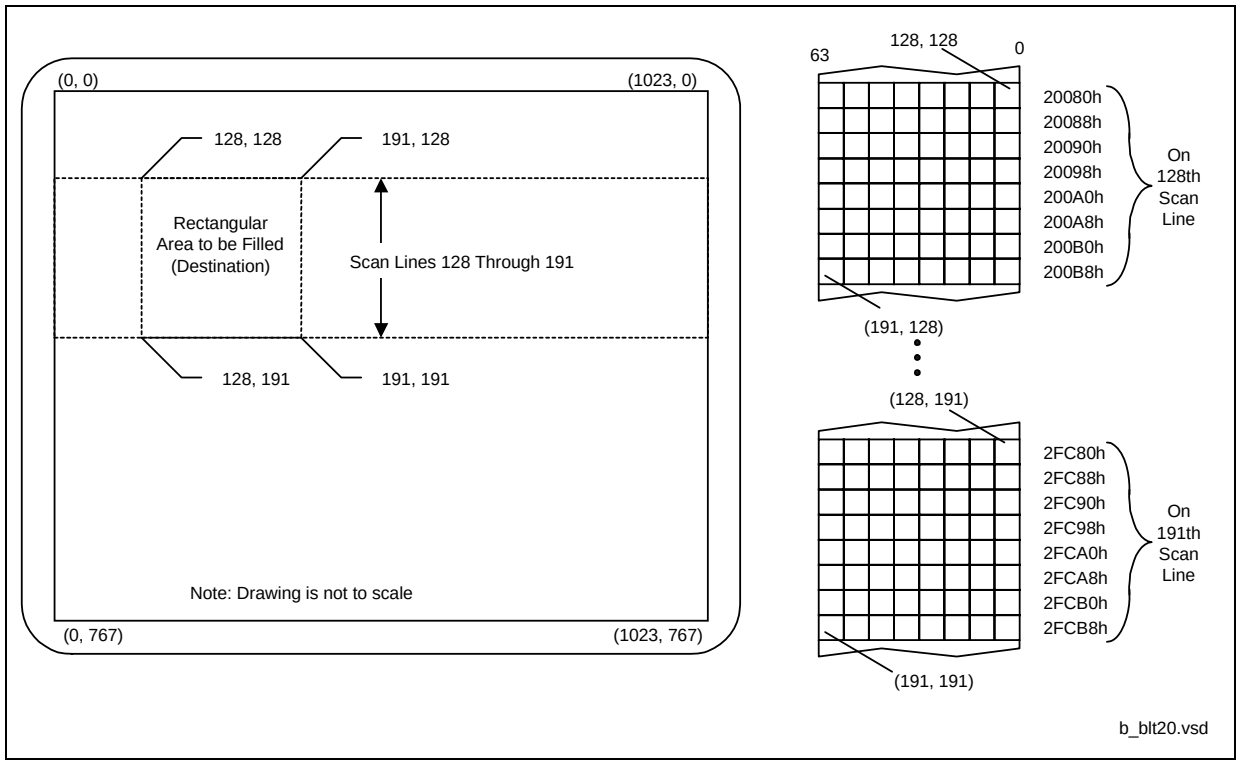
To accommodate discontinuous destination data, the Source and Destination Pitch Registers can be used to specify the offset in bytes from the beginning of one scan line’s worth of destination data to the next. Otherwise, if the destination data is contiguous, then an offset equal to the length of a scan line’s worth of destination data should be specified.

14.2.3 BLT Programming Examples

14.2.3.1 Pattern Fill — A Very Simple BLT

In this example, a rectangular area on the screen is to be filled with a color pattern stored as pattern data in off-screen memory. The screen has a resolution of 1024x768 and the graphics system has been set to a color depth of 8 bits per pixel.

Figure 14-12. On-Screen Destination for Example Pattern Fill BLT



As shown in the figure above, the rectangular area to be filled has its upper left-hand corner at coordinates (128, 128) and its lower right-hand corner at coordinates (191, 191). These coordinates define a rectangle covering 64 scan lines, each scan line's worth of which is 64 pixels in length — in other words, an array of 64x64 pixels. Presuming that the pixel at coordinates (0, 0) corresponds to the byte at address 00h in the frame buffer memory, the pixel at (128, 128) corresponds to the byte at address 20080h.

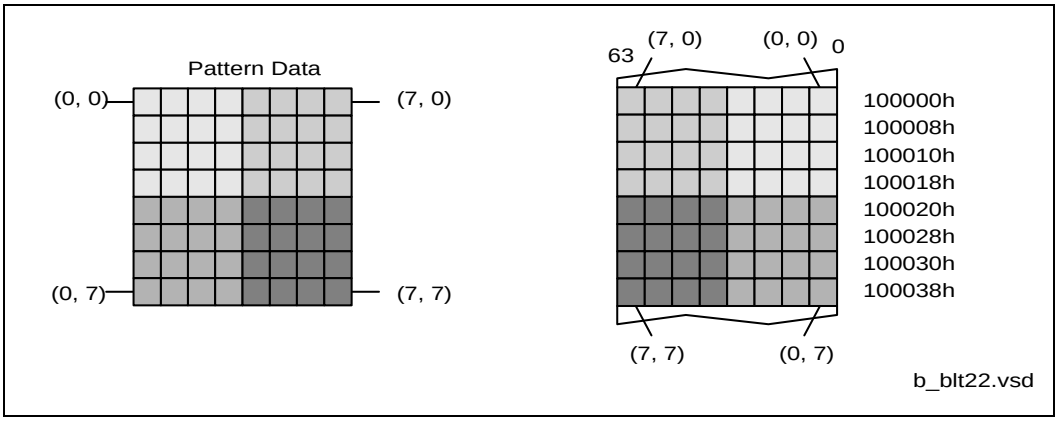


Figure 14-13. Pattern Data for Example Pattern Fill BLT

As shown in figure above, the pattern data occupies 64 bytes starting at address 100000h. As always, the pattern data represents an 8x8 array of pixels.

The BLT command packet is used to select the features to be used in this BLT operation, and must be programmed carefully. The vertical alignment field should be set to 0 to select the top-most horizontal row of the pattern as the starting row used in drawing the pattern starting with the top-most scan line covered by the destination. The pattern data is in color with a color depth of 8 bits per pixel, so the dynamic color enable should be asserted with the dynamic color depth field should be set to 0. Since this BLT operation does not use per-pixel write-masking (destination transparency mode), this field should be set to 0. Finally, the raster operation field should be programmed with the 8-bit value of F0h to select the bit-wise logical operation in which a simple copy of the pattern data to the destination takes place. Selecting this bit-wise operation in which no source data is used as an input causes the BLT engine to automatically forego either reading source data from the frame buffer.

The Destination Pitch Register must be programmed with number of bytes in the interval from the start of one scan line's worth of destination data to the next. Since the color depth is 8 bits per pixel and the horizontal resolution of the display is 1024, the value to be programmed into these bits is 400h, which is equal to the decimal value of 1024.

Bits [31:3] of the Pattern Address Register must be programmed with the address of the pattern data.

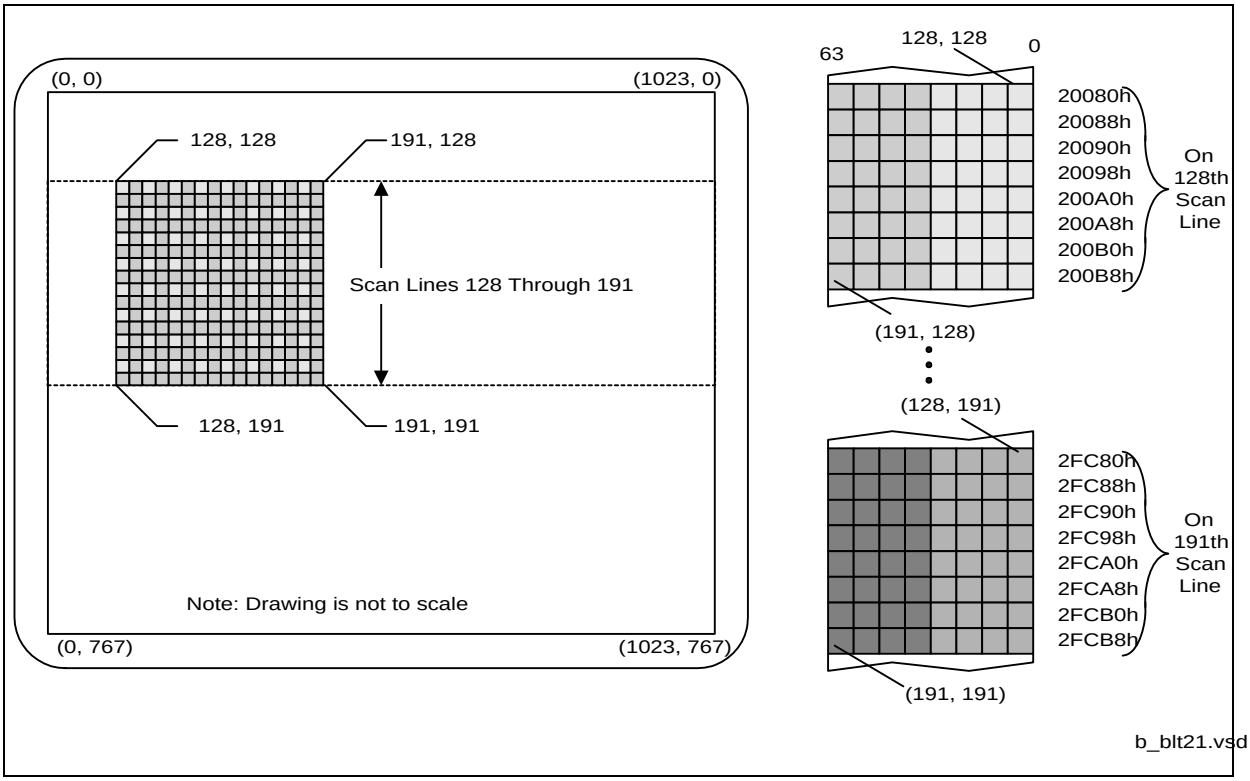
Similarly, bits [31:0] of the Destination Address Register must be programmed with the byte address at the destination that will be written to first. In this case, the address is 20080h, which corresponds to the byte representing the pixel at coordinates (128, 128).

This BLT operation does not use the values in the Source Address Register or the Source Expansion Background or Foreground Color Registers.

The Destination Width and Height Registers (or the Destination X and Y Coordinates) must be programmed with values that describe to the BLT engine the 64x64 pixel size of the destination location. The height should be set to carry the value of 40h, indicating that the destination location covers 64 scan lines. The width should be set to carry the value of 40h, indicating that each scan line's worth of destination data

occupies 64 bytes. All of this information is written to the ring buffer using the PAT_BLT (or XY_PAT_BLT) command packet.

Figure 14-14. Results of Example Pattern Fill BLT

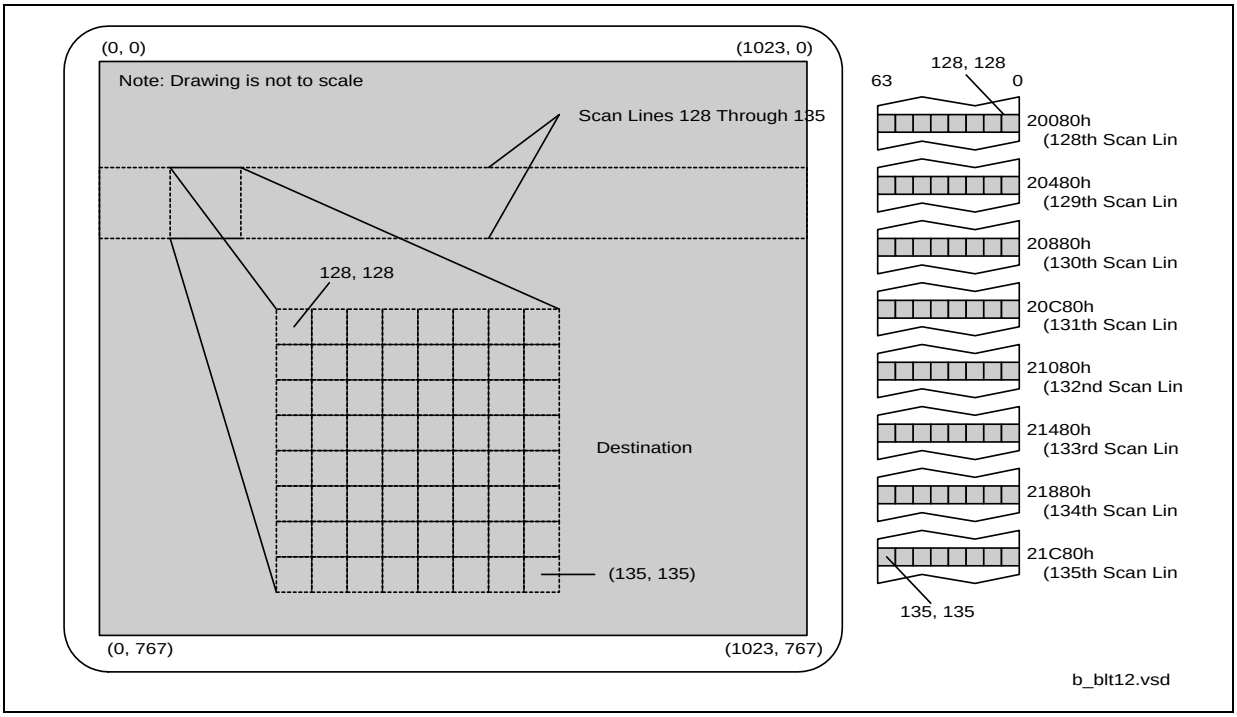


The figure above shows the end result of performing this BLT operation. The 8x8 pattern has been repeatedly copied (“tiled”) into the entire 64x64 area at the destination.

14.2.3.2 Drawing Characters Using a Font Stored in System Memory

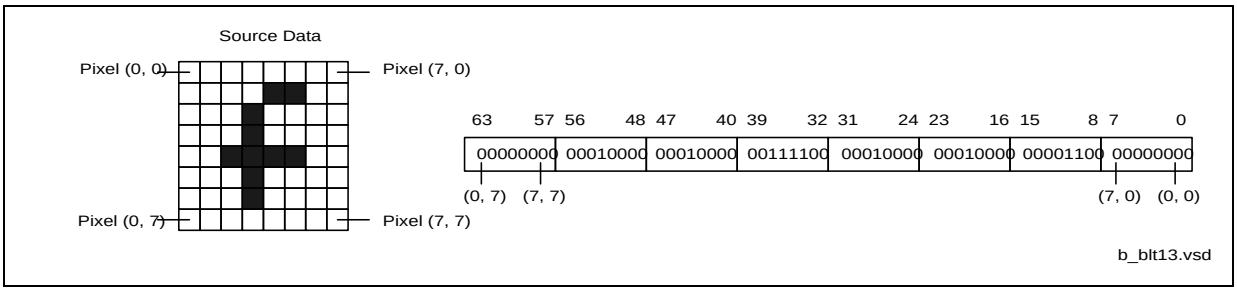
In this example BLT operation, a lowercase letter “f” is to be drawn in black on a display with a gray background. The resolution of the display is 1024x768, and the graphics system has been set to a color depth of 8 bits per pixel.

Figure 14-15. On-Screen Destination for Example Character Drawing BLT



The figure above shows the display on which this letter “f” is to be drawn. As shown in this figure, the entire display has been filled with a gray color. The letter “f” is to be drawn into an 8x8 region on the display with the upper left-hand corner at the coordinates (128, 128).

Figure 14-16. Source Data in System Memory for Example Character Drawing BLT



The figure above shows both the 8x8 pattern making up the letter “f” and how it is represented somewhere in the host’s system memory — the actual address in system memory is not important. The letter “f” is represented in system memory by a block of

monochrome graphics data that occupies 8 bytes. Each byte carries the 8 bits needed to represent the 8 pixels in each scan line's worth of this graphics data. This type of pattern is often used to store character fonts in system memory.

During this BLT operation, the host CPU will read this representation of the letter "f" from system memory, and write it to the BLT engine by performing memory writes to the ring buffer as an immediate monochrome BLT operand following the BLT_TEXT command. The BLT engine will receive this data through the command stream and use it as the source data for this BLT operation. The BLT engine will be set to the same color depth as the graphics system — 8 bits per pixel, in this case. Since the source data in this BLT operation is monochrome, color expansion must be used to convert it to an 8 bpp color depth. To ensure that the gray background behind this letter "f" is preserved, per-pixel write masking will be performed, using the monochrome source data as the pixel mask.

The BLT Setup and Text_immediate command packets are used to select the features to be used in this BLT operation. Only the fields required by these two command packets must be programmed carefully. The BLT engine ignores all other registers and fields. The source select field in the Text_immediate command must be set to 1, to indicate that the source data is provided by the host CPU through the command packet. Finally, the raster operation field should be programmed with the 8-bit value CCh to select the bit-wise logical operation that simply copies the source data to the destination. Selecting this bit-wise operation in which no pattern data is used as an input, causes the BLT engine to automatically forego reading pattern data from the frame buffer.

The Setup Pattern/Source Expansion Foreground Color Register to specify the color with which the letter "f" will be drawn. There is no Source address. All scan lines of the glyph are bit packed and the clipping is controlled by the ClipRect registers from the SETUP_BLT command and the Destination Y1, Y2, X1, and X2 registers in the TEXT_BLT command. Only the pixels that are within (inclusive comparisons) the clip rectangle are written to the destination surface.

The Destination Pitch Register must be programmed with a value equal to the number of bytes in the interval between the first bytes of each adjacent scan line's worth of destination data. Since the color depth is 8 bits per pixel and the horizontal resolution of the display is 1024 pixels, the value to be programmed into these bits is 400h, which is equal to the decimal value of 1024. Since the source data used in this BLT operation is monochrome, the BLT engine will not use a byte-oriented pitch value for the source data.

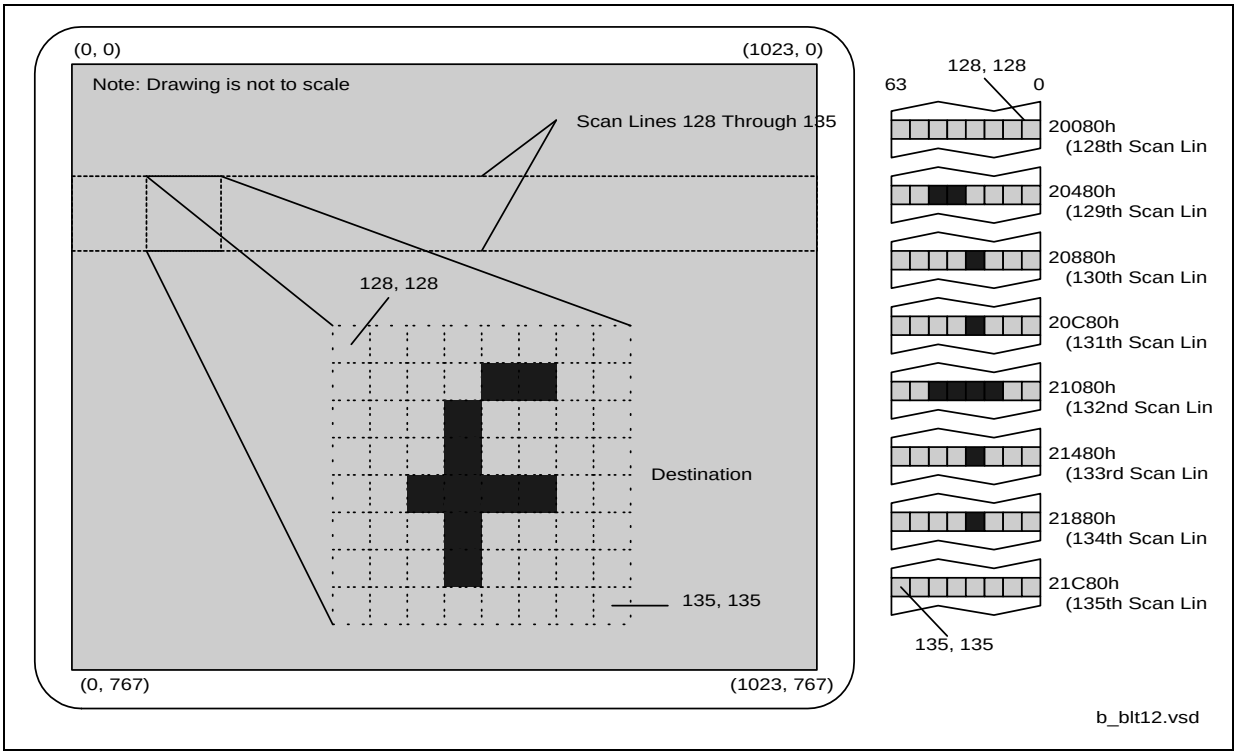
Since the source data is monochrome, color expansion is required to convert it to color with a color depth of 8 bits per pixel. Since the Setup Pattern/Source Expansion Foreground Color Register is selected to specify the foreground color of black to be used in drawing the letter "f", this register must be programmed with the value for that color. With the graphics system set for a color depth of 8 bits per pixel, the actual colors are specified in the RAMDAC palette, and the 8 bits stored in the frame buffer for each pixel actually specify the index used to select a color from that palette. This example assumes that the color specified at index 00h in the palette is black, and therefore bits [7:0] of this register should be set to 00h to select black as the foreground color. The BLT engine ignores bits [31:8] of this register because the selected color depth is 8 bits per pixel. Even though the color expansion being performed on the source data normally requires that both the foreground and background colors be specified, the value used to specify the background color is not important in this example. Per-pixel write-masking is being performed with the

monochrome source data as the pixel mask, which means that none of the pixels in the source data that will be converted to the background color will ever be written to the destination. Since these pixels will never be seen, the value programmed into the Pattern/Source Expansion Background Color Register to specify a background color is not important.

The Destination Width and Height Registers are not used. The Y1, Y2, X1, and X2 are used to describe to the BLT engine the 8x8 pixel size of the destination location. The Destination Y1 and Y2 address (or coordinate) registers must be programmed with the starting and ending scan line address (or Y coordinates) of the destination data. This address is specified as an offset from the start of the frame buffer of the scan line at the destination that will be written to first. The destination X1 and X2 registers must be programmed with the starting and ending pixel offsets from the beginning of the scan line.

This BLT operation does not use the values in the Pattern Address Register, the Source Expansion Background Color Register, or the Source Expansion Foreground Color Register.

Figure 14-17. Results of Example Character Drawing BLT



The preceding shows the end result of performing this BLT operation. Only the pixels that form part of the actual letter “f” have been drawn into the 8x8 destination location on the display, leaving the other pixels within the destination with their original gray color.

14.3 BLT Instruction Overview

This chapter defines the instructions used to control the 2D (BLT) rendering function.

The instructions detailed in this chapter are used across devices. However, slight changes may be present in some instructions (i.e., for features added or removed), or some instructions may be removed entirely. Refer to the *Device Dependencies* chapter for summary information regarding device-specific behaviors/interfaces/features.

The XY instructions offload the drivers by providing X and Y coordinates and taking care of the access directions for overlapping BLTs without fields specified by the driver.

Color pixel sizes supported are 8, 16, and 32 bits per pixel (bpp). All pixels are naturally aligned.

14.4 BLT Engine State

Most of the BLT instructions are state-free, which means that all states required to execute the command is within the instruction. If clipping is not used, then there is no shared state for many of the BLT instructions. This allows the BLT Engine to be shared by many drivers with minimal synchronization between the drivers.

Instructions which share state are:

- All instructions that are X,Y commands and use the Clipping Rectangle by asserting the Clip Enable field
- All XY_Setup Commands (XY_SETUP_BLT and XY_SETUP_MONO_PATTERN_SL_BLT) load the shared state for the following commands:
  - XY_PIXEL_BLT (Negative Stride (=Pitch) Not Allowed)
  - XY_SCANLINES_BLT
  - XY_TEXT_BLT (Negative Stride (=Pitch) Not Allowed)
  - XY_TEXT_IMMEDIATE_BLT (Negative Stride (=Pitch) Not Allowed)

State registers that are saved & restored in the Logical Context:

BR1+	Setup Control (Solid Pattern Select, Clipping Enable, Mono Source Transparency Mode, Mono Pattern Transparency Mode, Color Depth[1:0], Raster Operation[7:0], & Destination Pitch[15:0]) + 32bpp Channel Mask[1:0], Mono / Color Pattern
BR05	Setup Background Color
BR06	Setup Foreground Color
BR07	Setup Pattern Base Address
BR09	Setup Destination Base Address
BR20	DW0 for a Monochrome Pattern
BR21	DW1 for a Monochrome Pattern
BR24	ClipRectY1’X1
BR25	ClipRectY2’X2

14.5 Cacheable Memory Support

The BLT Engine can be used to transfer data between cacheable (“system”) memory and uncached (“main”, or “UC”) graphics memory using the BLT instructions. The GTT must be properly programmed to map memory pages as cacheable or UC. Only linear-mapped (not tiled) surfaces can be mapped as cacheable.

Transfers between cacheable sources and cacheable destinations are not supported. Patterns and monochrome sources cannot be located in cacheable memory.

Cacheable write operands do not snoop the processor’s cache nor update memory until evicted from the render cache. Cacheable read or write operands are not snooped (nor invalidated) from either internal cache by external (processor, hublink,…) accesses.

14.6 Device Cache Coherency: Render and Texture Caches

Software must initiate cache flushes to enforce coherency between the render and texture caches, i.e., both the render and texture caches must be flushed before a BLT destination surface can be reused as a texture source. Color sources and destinations use the render cache, while patterns and monochrome sources use the texture cache.

14.7 BLT Engine Instructions

The Instruction Target field is used as an opcode by the BLT Engine state machine to qualify the control bits that are relevant for executing the instruction. The descriptions for each DWord and bit field are contained in the *BLT Engine Instruction Field Definition* section. Each DWord field is described as a register, but none of these registers can be written of read through a memory mapped location – they are internal state only.

14.7.1 Blt Programming Restrictions

- **Overlapping Source/Destination BLTs:** The following condition must be avoided when programming the Blt engine: Linear surfaces with a cache line in scan line Y for the source stream overlapping with a cache line in scan line Y-1 for the dest stream (=> non-aligned surface pitches). The cache coherency rules combined with the Blitter data consumption rules result in UNDEFINED operation. (Note that this restriction will likely follow forward to future products due to architectural complexities.) There are two suggested software workarounds:
  - In order to perform coherent overlapping Blts, (a) the Source and Destination Base Address registers must hold the same value (without alignment restriction), and (b) the Source and Destination Pitch registers (BR11,BR13) must both be a multiple of **64 bytes**.
  - If (a) isn't possible, do overlapping source copy BLTs as two blits, using a separate intermediate surface.
- All reserved fields must be programmed to 0s.
- When using monosource or text data (bit/byte/word aligned): do not program pixel widths greater than 32,745 pixels.

14.8 Fill/Move Instructions

These instructions use linear addresses with width and height. BLT clipping is not supported.

14.8.1 COLOR_BLT (Fill)

COLOR_BLT is the simplest BLT operation. It performs a color fill to the destination (with a possible ROP). The only operand is the destination operand which is written dependent on the raster operation. The solid pattern color is stored in the pattern background register.

This instruction is optimized to run at the maximum memory write bandwidth.

The typical Raster operation code = F0 which performs a copy of the pattern background register to the destination.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	**Instruction Target (Opcode) :** 40h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:05	**Reserved.** Note no tiling specification allowed for this non-XY blit command. Only linear blits are allowed.
	04:00	**DWord Length:** 03h
1 = BR13	31:26	**Reserved.**
	25:24	**Color Depth:** 00 = 8 bit color. 01 = 16 bit color (656). 10 = 16 bit color (1555). 11 = 32 bit color
	23:16	**Raster Operation:**
	15:00	**Destination Pitch (signed):** Destination pitch in bytes (Same as before).
2 = BR14	31:16	**Destination Height (in scan lines):**
	15:00	**Destination Width (in bytes):**
3 = BR09	31:00	**Destination Address:** Address of the first byte to be written
4 = BR16	31:00	**Solid Pattern Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]

14.8.2 SRC_COPY_BLT (Move)

This BLT instruction performs a color source copy where the only operands involved is a color source and destination of the same bit width.

The source and destination operands may overlap. The command must indicate the horizontal and vertical directions: either forward or backwards to avoid data corruption. The X direction (horizontal) field applies to both the destination and source operands. The source and destination pitches (stride) are signed.

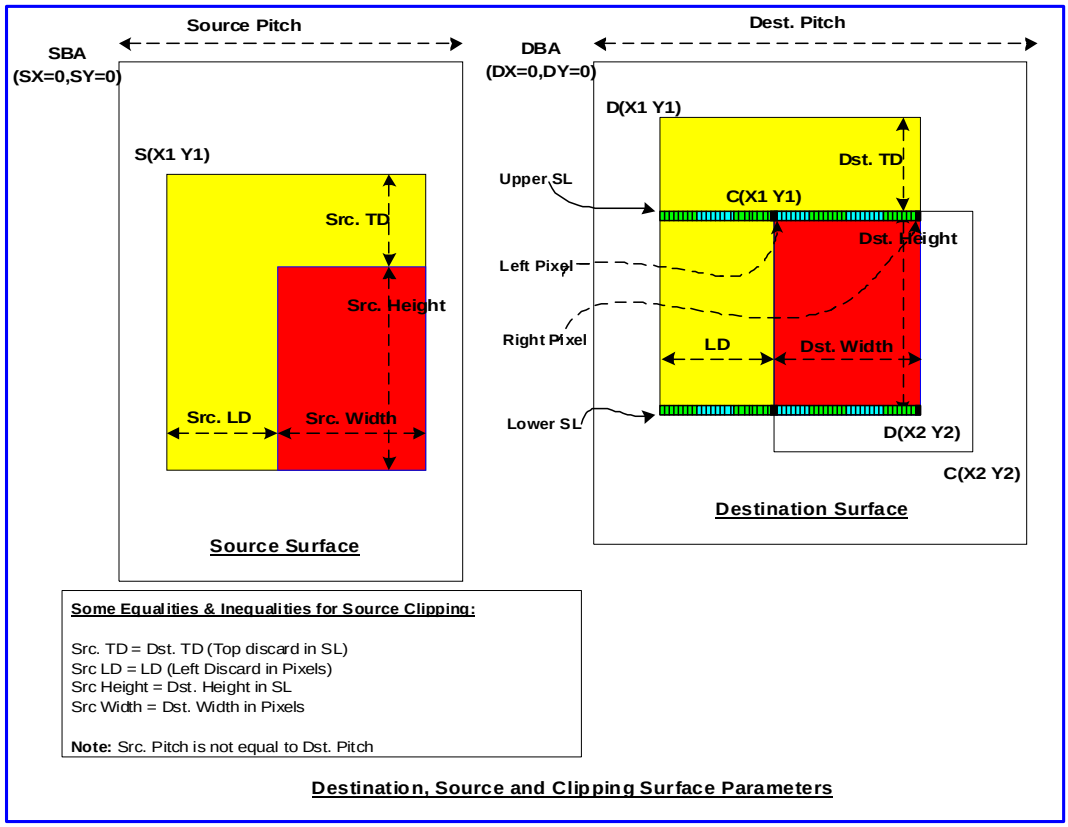
DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h – 2D Processor
	28:22	**Instruction Target (Opcode) :** 43h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:05	**Reserved.** Note no tiling specification allowed for this non-XY blit command. Only linear blits are allowed.
	04:00	**Dword Length:** 04h
1 = BR13	31	**Reserved.**
	30	**X Direction** (1 = written from right to left (decrementing = backwards); 0 = incrementing)
	29:26	**Reserved.**
	25:24	**Color Depth:** 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	**Raster Operation:**
	15:00	**Destination Pitch (signed):** Destination pitch in bytes (Same as before).
2 = BR14	31:16	**Destination Height (in scan lines):**
	15:00	**Destination Width (in bytes):**
3 = BR09	31:00	**Destination Address:** Address of the first byte to be written
	31:14	**Reserved.**
4 = BR11	15:00	**Source Pitch:** (double word aligned and signed)
5 = BR12	31:00	**Source Address:** Address of the first byte to be read.

14.9 2D (X,Y) BLT Instructions

Most BLT instructions (prefixed with “XY_”) use 2D X,Y coordinate specifications vs. lower-level linear addresses. These instructions also support simple 2D clipping against a clip rectangle.

The top and left Clipping coordinates are inclusive. The bottom and right coordinates are exclusive. The BLT Engine performs a trivial reject for all CLIP BLT instructions before performing any accesses.

Negative destination and source coordinates are supported. In the case of negative source coordinates, the destination X1 and Y1 are modified by the absolute value of the negative source coordinate before the destination clip checking and final drawing coordinates are calculated. The absolute value of the source negative coordinate is added to the corresponding destination coordinate. The BLT engine clipping also checks for (DX2 [or = DX1) or (DY2 [or = DY1) after this calculation and if true, then the BLT is totally rejected.



Destination, Source and Clipping Surface Parameters

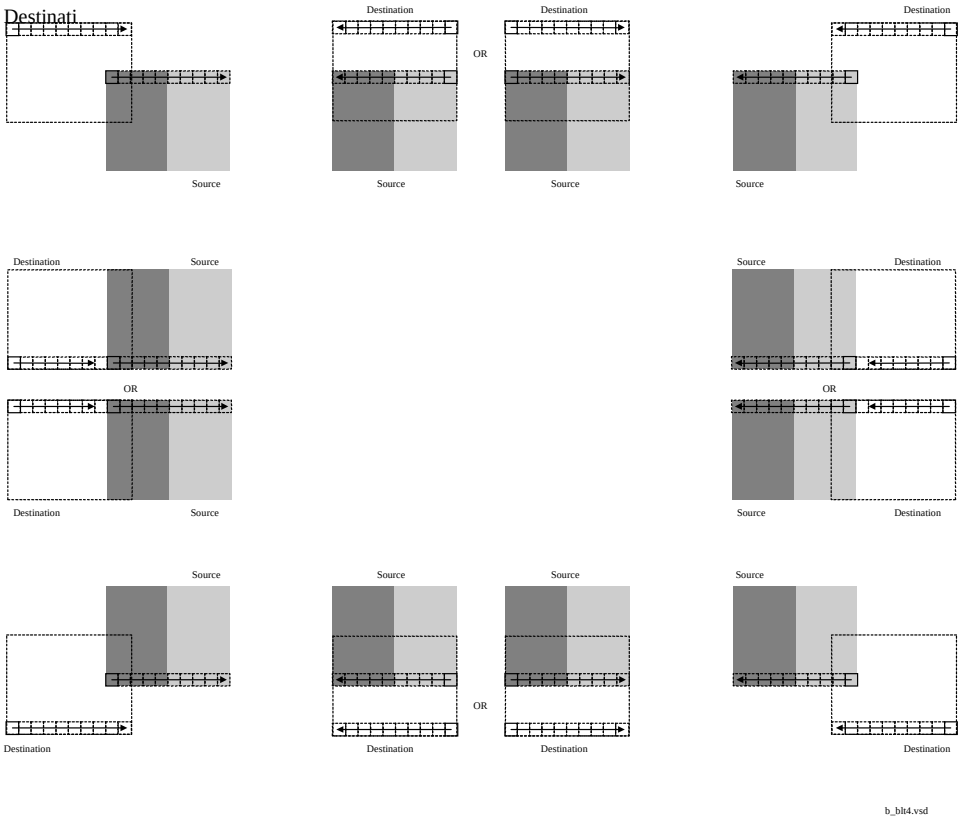
DX1, DY1, CX1, and CY1 are inclusive, while DX2, DY2, CX2, and CY2 are exclusive.

Destination pixel address = (Destination Base Address + (Destination Y coordinate * Destination pitch) + (Destination X coordinate * bytes per pixel)).

Source pixel address = (Source Base Address + (Source Y coordinate * Source pitch) + (Source X coordinate * bytes per pixel)).

Since there is 1 set of Clip Rectangle registers, the Interrupt Ring BLT commands either MUST NEVER enable clipping with these command and never use the XY_Pixel_BLT, XY_Scanline_BLT, nor XY_Text_BLT commands or it must use context switching. The Interrupt rings can also use the non-clipped, linear address commands specified before this section.

The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is **less than** Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is **less than** Destination Y1, then the scan line accesses are performed backwards.



14.9.1 XY_SETUP_BLT

This setup instruction supplies common setup information including clipping coordinates used by the XY commands: XY_PIXEL_BLT, XY_SCANLINE_BLT, XY_TEXT_BLT, and XY_TEXT_BLT_IMMEDIATE.

These are the only instructions that require that state be saved between instructions other than the Clipping parameters. There are 5 dedicated registers to contain the state for these 3 instructions. All other BLTs use a temporary version of these. The 5 double word registers are: DW1 (Setup Control), DW6 (Setup Foreground color), DW5 (Setup Background color), DW7 (Setup Pattern address), and DW4 (Setup Destination Base Address).

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 01h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:12	**Reserved.**
	11	**Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10: 08	**Reserved**
	07:00	**Dword Length:** 06h
1 = BR01	31	**Reserved.**
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29	**Mono Source Transparency Mode:** (1 = transparency enabled; 0 = use background)
	28:26	**Reserved.**
	25:24	**Color Depth:** All 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	**Raster Operation:**
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement (Negative Pitch Not allowed for Pixel nor Text) For Tiled surfaces (bit_11 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
2 = BR24	31:16	**ClipRect Y1 Coordinate (Top):** (30:16 = 15 bit positive number)
	15:00	**ClipRect X1 Coordinate (Left):** (14:00 = 15 bit positive number)
3 = BR25	31:16	**ClipRect Y2 Coordinate (Bottom):** (30:16 = 15 bit positive number)
	15:00	**ClipRect X2 Coordinate (Right):** (14:00 = 15 bit positive number)
4 = BR09	31:00	**Setup Destination Base Address:** (base address of the destination surface: X=0, Y=0) **When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.**
5 = BR05	31:00	**Setup Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0] All
6 = BR06	31:00	**Setup Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0] (SLB & TB only)
7 = BR07	31:00	**Setup Pattern Base Address for Color Pattern:** (26:06 are implemented) (SLB only) (Note no NPO2 change here). The pattern data must be located in linear memory.

14.9.2 XY_SETUP_MONO_PATTERN_SL_BLT

This setup instruction supplies common setup information including clipping coordinates used exclusively with the following instruction: XY_SCANLINE_BLT (SLB) - 1 scan line of monochrome pattern and destination are the only operands allowed.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 11h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:12	**Reserved.**
	11	**Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10: 08	**Reserved**
	07:00	**Dword Length:** 07h
1 = BR01	31	**Solid Pattern Select:** (1 = solid pattern; 0 = no solid pattern) - (SLB & Pixel only)
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29	**Reserved.**
	28	**Mono Pattern Transparency Mode:** (1 = transparency enabled; 0 = use background)
	27:26	**Reserved.**
	25:24	**Color Depth:** 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	**Raster Operation:**
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement (Negative Pitch Not allowed for Pixel nor Text) For Tiled surfaces (bit_11 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
2 = BR24	31:16	**ClipRect Y1 Coordinate (Top):** (30:16 = 15 bit positive number)
	15:00	**ClipRect X1 Coordinate (Left):** (14:00 = 15 bit positive number)
3 = BR25	31:16	**ClipRect Y2 Coordinate (Bottom):** (30:16 = 15 bit positive number)
	15:00	**ClipRect X2 Coordinate (Right):** (14:00 = 15 bit positive number)
4 = BR09	31:00	**Setup Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR05	31:00	**Setup Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
6 = BR06	31:00	**Setup Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 = BR20	31:00	**DW0 (least significant) for a Monochrome Pattern:**
8 = BR21	31:00	**DW1 (most significant) for a Monochrome Pattern:**

14.9.3 XY_SETUP_CLIP_ BLT

This command is used to only change the clip coordinate registers. These are the same clipping registers as the Setup clipping registers above.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 03h
	21:12	**Reserved.**
	11	**Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10: 08	**Reserved**
	07:00	**Dword Length:** 01h
1 = BR24	31:16	**ClipRect Y1 Coordinate (Top):** (30:16 = 15 bit positive number)
	15:00	**ClipRect X1 Coordinate (Left):** (14:00 = 15 bit positive number)
2 = BR25	31:16	**ClipRect Y2 Coordinate (Bottom):** (30:16 = 15 bit positive number)
	15:00	**ClipRect X2 Coordinate (Right):** (14:00 = 15 bit positive number)

14.9.4 XY_PIXEL_BLT

The Destination X coordinate and Destination Y coordinate is compared with the ClipRect registers. If it is within all 4 comparisons, then the pixel supplied in the XY_SETUP_BLT instruction is written with the raster operation to (Destination Y Address + (Destination Y coordinate * Destination pitch) + (Destination X coordinate * bytes per pixel)).

ROP field must specify pattern or fill with 0's or 1's. There is no source operand.

Negative Stride (= Pitch) specified in the Setup command is Not Allowed

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 24h
	21:12	**Reserved.**
	11	**Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10: 08	**Reserved**
	07:00	**Dword Length :** 00h
1 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)

14.9.5 XY_SCANLINES_BLT

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

Solid pattern should use the XY_SETUP_MONO_PATTERN_SL_BLT instruction.

ROP field must specify pattern or fill with 0's or 1's. There is no source operand.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 25h
	21:15	**Reserved.**
	14:12	**Pattern Horizontal Seed:** (pixel of the scan line to start on corresponding to DST X=0)
	11	**Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10:08	**Pattern Vertical Seed:** (scan line of the 8x8 pattern to start on corresponding to DST Y=0)
	07:00	**Dword Length:** 01h
1 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)
2 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)

14.9.6 XY_TEXT_BLT

All source scan lines and pixels that fall within the ClipRect Y and X coordinates are written. The source address corresponds to Destination X1 and Y1 coordinate.

Text is either bit or byte packed. Bit packed means that the next scan line starts 1 pixel after the end of the current scan line with no bit padding. Byte packed means that the next scan line starts on the first bit of the next byte boundary after the last bit of the current line.

Source expansion color registers are always in the SETUP_BLT.

Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 26h
	21:17	**Reserved.**
	16	**Bit (0) / Byte (1) packed:** Byte packed is for the NT driver
	15:12	**Reserved.**
	11	**Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10: 08	**Reserved**
	07:00	**Dword Length:** 02h
1 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)
2 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)
3 = BR12	31:00	**Source Address:** (address of the first byte on scan line corresponding to Dst X1,Y1) (Note no NPO2 change here)

14.9.7 XY_TEXT_IMMEDIATE_BLT

This instruction allows the Driver to send data through the instruction stream that eliminates the read latency of reading a source from memory. If an operand is in system cacheable memory and either small or only accessed once, it can be copied directly to the instruction stream versus to graphics accessible memory.

The IMMEDIATE_BLT data MUST transfer an even number of doublewords. The BLT engine will hang if it does not get an even number of doublewords.

All source scan lines and pixels that fall within the ClipRect X and Y coordinates are written. The source data corresponds to Destination X1 and Y1 coordinate.

Source expansion color registers are always in the SETUP_BLT.

NEGATIVE STRIDE (= PITCH) IS NOT ALLOWED.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h – 2D Processor
	28:22	**Instruction Target (Opcode):** 31h
	21:17	**Reserved.**
	16	**Bit (0) / Byte (1) packed:** Byte packed is for the NT driver
	15:12	**Reserved.**
	11	**Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10: 08	**Reserved**
	07:00	**Dword Length :** 01+ DWL = (Number of Immediate double words)h
1 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)
2 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)
3	31:00	**Immediate Data DW 0:**
4	31:00	**Immediate Data DW 1:**
5 thru DWL+3	S	**Immediate Data DWs 2 through DWORD_LENGTH (DWL):**

14.9.8 XY_COLOR_BLT

COLOR_BLT is the simplest BLT operation. It performs a color fill to the destination (with a possible ROP). The only operand is the destination operand which is written dependent on the raster operation. The solid pattern color is stored in the pattern background register.

This instruction is optimized to run at the maximum memory write bandwidth.

The typical (and fastest) Raster operation code = F0 which performs a copy of the pattern background register to the destination.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 50h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:12	**Reserved.**
	11	**Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10: 08	**Reserved**
	07:00	**Dword Length:** 04h
1 = BR13	31	**Reserved.**
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29:26	**Reserved.**
	25:24	**Color Depth:** 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	**Raster Operation:**
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)
3 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR16	31:00	**Solid Pattern Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]

14.9.9 XY_PAT_BLT

PAT_BLT is used when there is no source and the color pattern is not trivial (is not a solid color only).

If clipping is enabled, all scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 51h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:15	**Reserved.**
	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)
	11	**Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10:08	**Pattern Vertical Seed: (**Starting Scan line of the 8x8 pattern corresponding to DST Y=0)
	07:00	**Dword Length:** 04h
1 = BR13	31	**Reserved.**
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29:26	**Reserved.**
	25:24	**Color Depth:** 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	**Raster Operation:**
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)
3 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR15	31:00	**Pattern Base Address:** (28:06 are implemented) (Note no NPO2 change here). The pattern data must be located in linear memory.

14.9.10 XY_PAT_CHROMA_BLT

PAT_BLT is used when there is no source and the color pattern is not trivial (is not a solid color only).

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 76h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:17	**Transparency Range Mode:** (chroma-key) – Dst Chroma-key modes ONLY (SRC ILLEGAL)
	16:15	**Reserved.**
	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)
	11	**Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10:08	**Pattern Vertical Seed: (**Starting Scan line of the 8x8 pattern corresponding to DST Y=0)
	07:00	**Dword Length:** 06h
1 = BR13	31	**Reserved.**
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29:26	**Reserved.**
	25:24	**Color Depth:** 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	**Raster Operation:**
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)
3 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR15	31:00	**Pattern Base Address:** (26:06 are used, other bits are ignored) (Note no NPO2 change here). The pattern data must be located in linear memory.
6 = BR18	31:00	**Transparency Color Low:** (Chroma-key Low = Pixel Greater or Equal)
7 = BR19	31:00	**Transparency Color High:** (Chroma-key High = Pixel Less or Equal)

14.9.11 XY_PAT_BLT_IMMEDIATE

PAT_BLT_IMMEDIATE is used when there is no source and the color pattern is not trivial (is not a solid color only) and the pattern is pulled through the command stream. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64DWs) for 8, 16, and 32 bpp color patterns.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 72h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:15	**Reserved.**
	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)
	11	**Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10:08	**Pattern Vertical Seed:** (starting scan line of the 8x8 pattern corresponding to DST Y=0)
	07:00	**Dword Length:** 03+ DWL = (Number of Immediate double)h
1 = BR13	31	**Reserved**
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29:26	**Reserved.**
	25:24	**Color Depth:** 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	**Raster Operation:**
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)
3 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5	31:00	**Immediate Data DW 0:**
6	31:00	**Immediate Data DW 1:**
7 thru DWL+3	S	**Immediate Data DWs 2 through DWORD_LENGTH (DWL):**

14.9.12 XY_PAT_CHROMA_BLT_IMMEDIATE

PAT_BLT_IMMEDIATE is used when there is no source and the color pattern is not trivial (is not a solid color only) and the pattern is pulled through the command stream. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64DWs) for 8, 16, and 32 bpp color patterns.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 77h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:17	**Transparency Range Mode:** (chroma-key) – Dst Chroma-key modes ONLY (SRC ILLEGAL)
	16:15	**Reserved.**
	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)
	11	**Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10:08	**Pattern Vertical Seed:** (starting scan line of the 8x8 pattern corresponding to DST Y=0)
	07:00	**Dword Length:** 05+ DWL = (Number of Immediate double)h
1 = BR13	31	**Reserved.**
	30	**Clipping Enable** (1 = enabled; 0 = disabled)
	29:26	**Reserved.**
	25:24	**Color Depth:** 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	**Raster Operation:**
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)
3 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR18	31:00	**Transparency Color Low:** (Chroma-key Low = Pixel Greater or Equal)
6 = BR19	31:00	**Transparency Color High:** (Chroma-key High = Pixel Less or Equal)
7	31:00	**Immediate Data DW 0:**
8	31:00	**Immediate Data DW 1:**
9 thru DWL+3	S	**Immediate Data DWs 2 through DWORD_LENGTH (DWL):**

14.9.13 XY_MONO_PAT_BLT

MONO_PAT_BLT is used when we have no source and the monochrome pattern is not trivial (is not a solid color only). The monochrome pattern is loaded from the instruction stream.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the pattern is 0. When the pattern bit is 1, then the pattern foreground color is used in the ROP operation.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	**Instruction Target (Opcode) :** 52h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:15	**Reserved.**
	14:12	**Pattern Horizontal Seed:** (pixel of the scan line to start on corresponding to DST X=0)
	11	**Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10:08	**Pattern Vertical Seed:** (starting scan line of the 8x8 pattern corresponding to DST Y=0)
	07:00	**Dword Length:** 07h
1 = BR13	31	**Reserved.**
	30	**Clipping Enable** (1 = enabled; 0 = disabled)
	29	**Reserved.**
	28	**Mono Pattern Transparency Mode:** (1 = transparency enabled; 0 = use background)
	27:26	**Reserved.**
	25:24	**Color Depth:** 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	**Raster Operation:**
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** 31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)

DWord	Bit	Description
3 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR16	31:00	**Pattern Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
6 = BR17	31:00	**Pattern Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 = BR20	31:00	**Pattern Data 0:** (least significant DW)
8 = BR21	31:00	**Pattern Data 1:** (most significant DW)

14.9.14 XY_MONO_PAT_FIXED_BLT

MONO_PAT_FIXED_BLT is used when we have no source and the monochrome pattern is not trivial (is not a solid color only). The monochrome pattern is one of 10 fixed patterns described below. The pattern seeds can still be used with the fixed patterns, creating even more fixed patterns. This eliminates 2 doublewords compared to the XY_MONO_PAT_BLT command packet.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

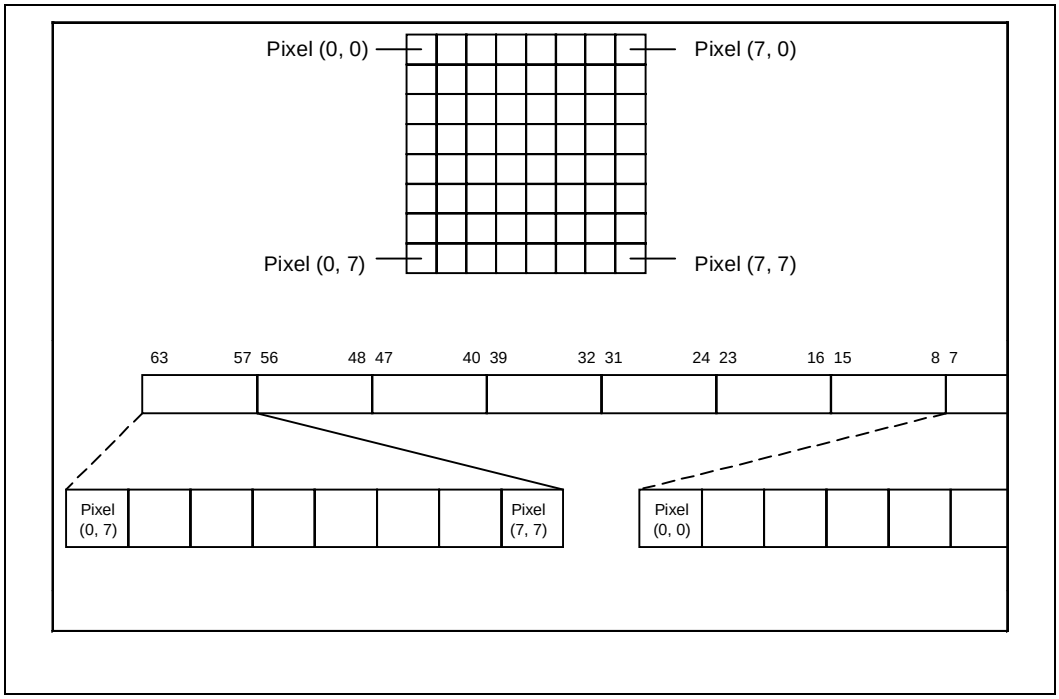
The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the pattern is 0. When the pattern bit is 1, then the pattern foreground color is used in the ROP operation.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	**Instruction Target** (Opcode)**:** 59h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19	**Reserved.**
	18:15	**Fixed Pattern:** 0000 HS_HORIZONTAL 0001 HS_VERTICAL 0010 HS_FDIAGONAL 0011 HS_BDIAGONAL 0100 HS_CROSS 0101 HS_DIAGCROSS 0110 Reserved 0111 Reserved 1000 Screen Door 1001 SD Wide 1010 Walking Bit (one) 1011 Walking Zero 1100 Reserved 1101 Reserved 1110 Reserved
	14:12	**Pattern Horizontal Seed:** (pixel of the scan line to start on corresponding to DST X=0)
	11	**Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10:08	**Pattern Vertical Seed:** (starting scan line of the 8x8 pattern corresponding to DST Y=0)
	07:00	**Dword Length:** 05h
1 = BR13	31	**Reserved.**

DWord	Bit	Description
	30	**Clipping Enable** (1 = enabled; 0 = disabled)
	29	**Reserved.**
	28	**Mono Pattern Transparency Mode:** (1 = transparency enabled; 0 = use background)
	27	**Bit Mask Enable:** (1 = use bit mask register for bit writes; 0 = disabled)
	27:26	**Reserved.**
	25:24	**Color Depth:** 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	**Raster Operation:**
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)
3 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR16	31:00	**Pattern Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
6 = BR17	31:00	**Pattern Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]

14.9.14.1 Monochrome Pattern Memory Format

The monochrome pattern is made of 8 bytes that correspond to the 8 pixels per scan line and 8 scan lines. Byte 0 corresponds to scan line 0, byte 1 corresponds to scan line 1,…, and byte 7 corresponds to scan line 7. The bits within each byte are transposed. Pixel 0 is bit 7, pixel 1 is bit 6,…, pixel 7 is bit 0. The diagram below illustrates the byte and bit relationship to the pixels of the pattern.



14.9.14.2 HS_HORIZONTAL 0

Bit 7 0,0							0 7,0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

14.9.14.3 HS_VERTICAL 1

Bit 7 0,0							0 7,0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0

14.9.14.4 HS_FDIAGONAL 2

Bit 7 0,0							0 7,0
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

14.9.14.5 HS_BDIAGONAL 3

Bit 7 0,0							0 7,0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0

14.9.14.6 HS_CROSS 4

Bit 7 0

0,0 7,0

0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
1	1	1	1	1	1	1	1
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0

14.9.14.7 HS_DIAGCROSS 5

Bit 7 0

0,0 7,0

1	0	0	0	0	0	0	1
0	1	0	0	0	0	1	0
0	0	1	0	0	1	0	0
0	0	0	1	1	0	0	0
0	0	0	1	1	0	0	0
0	0	1	0	0	1	0	0
0	1	0	0	0	0	1	0
1	0	0	0	0	0	0	1

14.9.14.8 Screen Door 8

Bit 7 0

0,0 7,0

0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0

14.9.14.9 SD Wide 9

Bit 7 0

0,0 7,0

1	1	0	0	1	1	0	0
0	0	1	1	0	0	1	1
1	1	0	0	1	1	0	0
0	0	1	1	0	0	1	1
1	1	0	0	1	1	0	0
0	0	1	1	0	0	1	1
1	1	0	0	1	1	0	0
0	0	1	1	0	0	1	1

14.9.14.10 Walking Bit (One) A

Bit 7 0,0							0 7,0
1	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	0
0	0	0	1	0	0	0	1
1	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	0
0	0	0	1	0	0	0	1

14.9.14.11 Walking Zero B

Bit 7 0,0							0 7,0
0	1	1	1	0	1	1	1
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	0	1	1	1	0
0	1	1	1	0	1	1	1
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	0	1	1	1	0

14.9.15 XY_SRC_COPY_BLT

This BLT instruction performs a color source copy where the only operands involved is a color source and destination of the same bit width.

The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is **less than** Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is **less than** Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.

The ROP value chosen must involve source and no pattern data in the ROP operation.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 53h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:16	**Reserved.**

DWord	Bit	Description
	15	**Src Tiling Enable:** 0 = Tiling Disabled (Linear) 1 = Tiling enabled (Tile-X only)
	14:12	**Reserved**
	11	**Dest Tiling Enable:** **0 = Tiling Disabled (Linear)** **1 = Tiling enabled (Tile-X only)**
	10: 8	**Reserved**
	7:0	**Dword Length:** 06h
1 = BR13	31	**Reserved.**
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29:26	**Reserved.**
	25:24	**Color Depth:** 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	**Raster Operation:**
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement **For Tiled Dest (bit 11 enabled) this ptich is of 512Byte granularity and can be up to 128Kbytes (or 32KDwords).**
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)
3 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Dest Tiling is enabled (Bit 11 enabled), this address is limited to 4Kbytes.
5 = BR26	31:16	**Source Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Source X1 Coordinate (Left):** (15:00 = 16 bit signed number)
6 = BR11	31:16	**Reserved**
	15:00	**Source Pitch (double word aligned) and in DWords:** [15:00] 2's complement. For Tiled Src (bit 15 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
7 = BR12	31:00	**Source Base Address:** (base address of the source surface: X=0, Y=0) When Src Tiling is enabled (Bit 15 enabled), this address is limited to 4Kbytes.

14.9.16 XY_SRC_COPY_CHROMA_BLT

This BLT instruction performs a color source copy with chroma-keying where the only operands involved is a color source and destination of the same bit width.

The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is **less than** Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is **less than** Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.

The ROP value chosen must involve source and no pattern data in the ROP operation.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 73h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:17	**Transparency Range Mode:** (chroma-key)
	16	**Reserved**
	15	**Src Tiling Enable:** 0 = Tiling Disabled (Linear) 1 = Tiling enabled (Tile-X only)
	14:12	**Reserved**
	11	**Dest Tiling Enable:** 0 = Tiling Disabled (Linear) 1 = Tiling enabled (Tile-X only)
	10: 08	**Reserved**
	07:00	**Dword Length:** 08h
1 = BR13	31	**Reserved.**
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29:26	**Reserved.**
	25:24	**Color Depth:** 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	**Raster Operation:**

DWord	Bit	Description
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement For Tiled Dest (bit 11 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)
3 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Dest Tiling is enabled (Bit 11 enabled), this address is limited to 4Kbytes.
5 = BR26	31:16	**Source Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Source X1 Coordinate (Left):** (15:00 = 16 bit signed number)
6 = BR11	31:16	**Reserved.**
	15:00	**Source Pitch (double word aligned) and in DWords:** [15:00] 2's complement. For Tiled Src (bit 15 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
7 = BR12	31:00	**Source Base Address:** (base address of the source surface: X=0, Y=0) When Src Tiling is enabled (Bit 15 enabled), this address is limited to 4Kbytes.
8 = BR18	31:00	**Transparency Color Low:** (Chroma-key Low = Pixel Greater or Equal)
9 = BR19	31:00	**Transparency Color High:** (Chroma-key High = Pixel Less or Equal)

14.9.17 XY_MONO_SRC_COPY_BLT

This BLT instruction performs a monochrome source copy where the only operands involved is a monochrome source and destination. The source and destination operands cannot overlap therefore the X and Y directions are always forward.

All non-text monochrome sources are word aligned. At the end of a scan line of monochrome source, all bits until the next word boundary are ignored. The monochrome source data bit position field [2:0] indicates the bit position within the first byte of the scan line that should be used as the first source pixel which corresponds to the destination X1 coordinate.

The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation. The ROP value chosen must involve source and no pattern data in the ROP operation. Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 54h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:17	**Monochrome source data bit position of the first pixel within a byte per scan line.**
	16:12	**Reserved.**
	11	**Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10: 08	**Reserved**
	07:00	**Doubleword Length:** 06h
1 = BR13	31	**Reserved.**
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29	**Mono Source Transparency Mode:** (1 = transparency enabled; 0 = use background)
	28:26	**Reserved.**
	25:24	**Color Depth:** 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	**Raster Operation:**
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)
3 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR12	31:00	**Source Address:** (address corresponding to DST X1,Y1) (Note no NPO2 change here)
6 = BR18	31:00	**Source Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 = BR19	31:00	**Source Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]

14.9.18 XY_MONO_SRC_COPY_ IMMEDIATE_BLT

This instruction allows the Driver to send monochrome data through the instruction stream, eliminating the read latency of the source during command execution.

The IMMEDIATE_BLT data MUST transfer an even number of doublewords and the exact number of quadwords.

All non-text monochrome sources are word aligned. At the end of a scan line of monochrome source, all bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates the bit position within the first byte of the scan line that should be used as the first source pixel which corresponds to the destination X1 coordinate.

The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation. The ROP value chosen must involve source and no pattern data in the ROP operation.

The monochrome source data supplied corresponds to the Destination X1 and Y1 coordinates.

Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 71h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:17	**Monochrome source data bit position of the first pixel within a byte per scan line.**
	16:12	**Reserved.**
	11	**Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10: 08	**Reserved**
	07:00	**Dword Length:** 05+ DWL = (Number of Immediate double words)h
1 = BR13	31	**Reserved.**
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29	**Mono Source Transparency Mode:** (1 = transparency enabled; 0 = use background)
	28:26	**Reserved.**
	25:24	**Color Depth:** 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	**Raster Operation:**

DWord	Bit	Description
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)
3 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR18	31:00	**Source Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
6 = BR19	31:00	**Source Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7	31:00	**Immediate Data DW 0:**
8	31:00	**Immediate Data DW 1:**
9 thru DWL+4	S	**Immediate Data DWs 2 through DWORD_LENGTH (DWL):**

14.9.19 XY_FULL_BLT

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source and pattern operands are the same bit width as the destination operand.

The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is **less than** Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is **less than** Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 = BR00	31:29	**Client:** 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 55h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:16	**Reserved.**
	15	**Src Tiling Enable:** 0 = Tiling Disabled (Linear) 1 = Tiling enabled (Tile-X only)
	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)
	11	**Dest Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10:08	**Pattern Vertical Seed:** (starting scan line of the 8x8 pattern corresponding to DST Y=0)
	07:00	**Doubleword Length:** 07h
1 = BR13	31	**Reserved.**
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29:26	**Reserved.**

DWord	Bit	Description
	25:24	**Color Depth:** 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	**Raster Operation:**
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement For Tiled Dest (bit 11 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)
3 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Dest Tiling is enabled (Bit 11 enabled), this address is limited to 4Kbytes.
5 = BR11	31:16	**Reserved.**
	15:00	**Source Pitch (double word aligned and signed) and in DWords:** [15:00] 2's complement. For Tiled Src (bit 15 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
6 = BR26	31:16	**Source Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Source X1 Coordinate (Left):** (15:00 = 16 bit signed number)
7 = BR12	31:00	**Source Base Address:** (base address of the source surface: X=0, Y=0) When Src Tiling is enabled (Bit 15 enabled), this address is limited to 4Kbytes.
8 = BR15	31:00	**Pattern Base Address:** (28:06 are implemented) (Note no NPO2 change here). The pattern data must be located in linear memory.

14.9.20 XY_FULL_IMMEDIATE_PATTERN_BLT

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source and immediate pattern operands are the same bit width as the destination operand. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64 DWs) for 8, 16, and 32 bpp color patterns.

The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is **less than** Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is **less than** Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 = BR00	31:29	**Client**: 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 74h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:16	**Reserved.**
	15	**Src Tiling Enable:** 0 = Tiling Disabled (Linear) 1 = Tiling enabled (Tile-X only)
	14:12	**Pattern Horizontal Seed:** (pixel of the scan line to start on corresponding to DST X=0)
	11	**Dest Tiling Enable:** 0 = Tiling Disabled (Linear) 1 = Tiling enabled (Tile-X only)
	10:8	**Pattern Vertical Seed:** (starting scan line of the 8x8 pattern corresponding to DST Y=0)
	7:0	**Doubleword Length:** 06+ DWL = (Number of Immediate double words)h
1 = BR13	31	**Reserved.**
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29:26	**Reserved.**

DWord	Bit	Description
	25:24	**Color Depth:** 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	**Raster Operation:**
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement For Tiled Dest (bit 11 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)
3 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Dest Tiling is enabled (Bit 11 enabled), this address is limited to 4Kbytes.
5 = BR11	31:16	**Reserved.**
	15:00	**Source Pitch (double word aligned and signed) and in DWords:** [15:00] 2's complement. For Tiled Src (bit 15 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
6 = BR26	31:16	**Source Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Source X1 Coordinate (Left):** (15:00 = 16 bit signed number)
7 = BR12	31:00	**Source Base Address:** (base address of the source surface: X=0, Y=0) When Src Tiling is enabled (Bit 15 enabled), this address is limited to 4Kbytes.
8	31:00	**Immediate Data DW 0:**
9	31:00	**Immediate Data DW 1:**
A thru DWL+4	S	**Immediate Data DWs 2 through DWORD_LENGTH (DWL):**

14.9.21 XY_FULL_MONO_SRC_BLT

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source operand is monochrome and the pattern operand is the same bit width as the destination.

The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation.

All non-text and non-immediate monochrome sources are word aligned. At the end of a scan line the monochrome source, the remaining bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates which bit position within the first byte should be used as the first source pixel which corresponds to the Destination X1 coordinate.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

Negative Stride (= Pitch) is NOT ALLOWED

DWord	Bit	Description
0 = BR00	31:29	**Client**: 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 56h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:17	**Monochrome source data bit position of the first pixel within a byte per scan line.**
	16:15	**Reserved.**
	14:12	**Pattern Horizontal Seed:** (pixel of the scan line to start on corresponding to DST X=0)
	11	**Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10:08	**Pattern Vertical Seed:** (starting address of the 8x8 pattern corresponding to DST Y=0)
	07:00	**Doubleword Length :** 07h
1 = BR13	31	**Reserved.**
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29	**Mono Source Transparency Mode:** (1 = transparency enabled; 0 = use background)
	28:27	**Reserved.**
	26	**Reserved.**

DWord	Bit	Description
	25:24	**Color Depth:** 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	**Raster operation:**
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)
3 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR12	31:00	**Mono Source Address:** (address corresponds to DST X1, Y1) (Note no NPO2 change here)
6 = BR18	31:00	**Source Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 = BR19	31:00	**Source Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
8 = BR15	31:00	**Pattern Base Address:** (28:06 are implemented **)** (Note no NPO2 change here). The pattern data must be located in linear memory.

14.9.22 XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source operand is a monochrome and the immediate pattern operand is the same bit width as the destination. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64DWs) for 8, 16, and 32 bpp color patterns.

The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation.

All non-text monochrome sources are word aligned. At the end of a scan line the monochrome source, the remaining bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates which bit position within the first byte should be used as the first source pixel which corresponds to the destination X1 coordinate.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 = BR00	31:29	**Client**: 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 75h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:17	**Monochrome source data bit position of the first pixel within a byte per scan line.**
	16:15	**Reserved.**
	14:12	**Pattern Horizontal Seed:** (pixel of the scan line to start on corresponding to DST X=0)
	11	**Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10:08	**Pattern Vertical Seed:** (starting address of the 8x8 pattern corresponding to DST Y=0)
	07:00	**Doubleword Length :** 06+ DWL = (Number of Immediate double words)h
1 = BR13	31	**Reserved.**
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29	**Mono Source Transparency Mode:** (1 = transparency enabled; 0 = use background)

DWord	Bit	Description
	28:26	**Reserved.**
	25:24	**Color Depth:** 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	**Raster operation:**
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)
3 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR12	31:00	**Mono Source Address:** (address corresponds to DST X1, Y1) (Note no NPO2 change here)
6 = BR18	31:00	**Source Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 = BR19	31:00	**Source Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
8	31:00	**Immediate Data DW 0:**
9	31:00	**Immediate Data DW 1:**
A thru DWL+4	S	**Immediate Data DWs 2 through DWORD_LENGTH (DWL):**

14.9.23 XY_FULL_MONO_PATTERN_BLT

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The pattern operand is monochrome and the source operand is the same bit width as the destination operand.

The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is **less than** Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is **less than** Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.

The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the pattern foreground color is used in the ROP operation.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

Setting both Solid Pattern Select =1 & Mono Pattern Transparency = 1 is mutually exclusive. The device implementation results in NO PIXELs DRAWN.

DWord	Bit	Description
0 = BR00	31:29	**Client**: 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 57h
	21:20	**32 bpp byte mask:** (21 =1= write alpha channel; 20=1= write RGB channels)
	19:16	**Reserved.**
	15	**Src Tiling Enable:** 0 = Tiling Disabled (Linear) 1 = Tiling enabled (Tile-X only)
	14:12	**Pattern Horizontal Seed:** (pixel of the scan line to start on corresponding to DST X=0)
	11	**Dest Tiling Enable:** 0 = Tiling Disabled (Linear) 1 = Tiling enabled (Tile-X only)
	10:08	**Pattern Vertical Seed:** (starting scan line of the 8x8 pattern corresponding to DST Y=0)
	07:00	**Dword Length :** 0Ah

DWord	Bit	Description
1 = BR13	31	**Solid Pattern Select:** (1 = solid pattern; 0 = no solid pattern)
	30	**Clipping Enable:** (1 = enabled; 0 = disabled)
	29	**Reserved.**
	28:27	**Mono Pattern Transparency Mode:** (1 = transparency enabled; 0 = use background)
	26	**Reserved.**
	25:24	**Color Depth:** 00 = 8 bit color 01 = 16 bit color 10 = 16 bit color (1555) 11 = 32 bit color (565)
	23:16	**Raster Operation:**
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement For Tiled Dest (bit 11 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)
3 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Dest Tiling is enabled (Bit 11 enabled), this address is limited to 4Kbytes.
5 = BR11	31:16	**Reserved.**
	15:00	**Source Pitch (double word aligned and signed) and in DWords:** [15:00] 2's complement. For Tiled Src (bit 15 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
6 = BR26	31:16	**Source Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Source X1 Coordinate (Left):** (15:00 = 16 bit signed number)
7 = BR12	31:00	**Source Base Address:** (base address of the source surface: X=0, Y=0) When Src Tiling is enabled (Bit 15 enabled), this address is limited to 4Kbytes.
8 = BR16	31:00	**Pattern Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
9 = BR17	31:00	**Pattern Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
A = BR20	31:00	**Pattern Data 0:** (least significant DW)
B = BR21	31:00	**Pattern Data 1:** (most significant DW)

14.9.24 XY_FULL_MONO_PATTERN_MONO_SRC_BLT

The full BLT provides the ability to specify all 3 operands: destination, source, and pattern. The pattern and source operands are monochrome.

The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation.

All non-text monochrome sources are word aligned. At the end of a scan line the monochrome source, the remaining bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates which bit position within the first byte should be used as the first source pixel which corresponds to the destination X1 coordinate.

The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the pattern is 0. When the source bit is 1, then the pattern foreground color is used in the ROP operation. The monochrome source transparency mode works identical to the pattern transparency mode.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

Setting both Solid Pattern Select =1 & Mono Pattern Transparency = 1 is mutually exclusive. The device implementation results in NO PIXELs DRAWN.

Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 = BR00	31:29	**Client**: 02h - 2D Processor
	28:22	**Instruction Target (Opcode):** 58h
	21:20	**32 bpp byte mask:** (21 = 1 = write alpha channel; 20 = 1 = write RGB channels)
	19:17	**Monochrome source data bit position of the first pixel within a byte per scan line.**
	16:15	**Reserved.**
	14:12	**Pattern Horizontal Seed:** (pixel of the scan line to start on corresponding to DST X = 0)
	11	**Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only)
	10:08	**Pattern Vertical Seed:** (starting scan line of the 8x8 pattern corresponding to DST Y = 0)
	07:00	**Doubleword Length :** 0Ah

DWord	Bit	Description
1 = BR13	31	**Solid Pattern Select:** (1 = solid pattern; 0 = no solid pattern)
	30	**Clipping Enable** (1 = enabled; 0 = disabled)
	29	**Mono Source Transparency Mode:** (1 = transparency enabled; 0 = use background)
	28	**Mono Pattern Transparency Mode:** (1 = transparency enabled; 0 = use background)
	27:26	**Reserved.**
	25:24	**Color Depth:** 00 = 8 bit color 01 = 16 bit color (565) 10 = 16 bit color (1555) 11 = 32 bit color
	23:16	**Raster Operation:**
	15:00	**Destination Pitch in DWords:** [15:00] 2's complement For Tiled surfaces (bit_11 enabled) this ptich is of 512Byte granularity and can be upto 128Kbytes (or 32KDwords).
2 = BR22	31:16	**Destination Y1 Coordinate (Top):** (31:16 = 16 bit signed number)
	15:00	**Destination X1 Coordinate (Left):** (15:00 = 16 bit signed number)
3 = BR23	31:16	**Destination Y2 Coordinate (Bottom):** (31:16 = 16 bit signed number)
	15:00	**Destination X2 Coordinate (Right):** (15:00 = 16 bit signed number)
4 = BR09	31:00	**Destination Base Address:** (base address of the destination surface: X=0, Y=0) When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.
5 = BR12	31:00	**Source Address:** (address corresponding to Dst X1,Y1) (Note no NPO2 change here)
6 = BR18	31:00	**Source Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 = BR19	31:00	**Source Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
8 = BR16	31:00	**Pattern Background Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
9 = BR17	31:00	**Pattern Foreground Color:** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
A =BR20	31:00	**Pattern Data 0:** (least significant DW)
B =BR21	31:00	**Pattern Data 1:** (most significant DW)

14.10 BLT Engine Instruction Field Definitions

This section describes the BLT Engine instruction fields. These descriptions are in the format of register descriptions. These registers are internal and are not readable. Some of these registers are state that is saved and restored for supporting separate software threads.

14.10.1 BR00—BLT Opcode & Control

Memory Offset Address: none
Default: 0000 0000
Attributes: not accessible

BR00 is the last executed instruction DWord 0. Bits [22:5] are written by every DW0 of every instruction. Bits [31:30] and [4:0] are status bits. Bits [28:27] are written from the DW0 [15:14] of a Setup instruction and Bit 29 is written with a 1 when ever a Setup instruction is written. Bit 29 is a decode of the Setup instruction Opcode.

31	30	29	28 … 24
Rsvd	Clip Inst	Setup Mono Pattern	Instruction Target (Opcode)

23 … 22	21 … 20	19 … 17	16
Instruction Target (Opcode)	32 bpp byte mask	Monochrome Source Start	Bit (0) / Byte (1) Packed

15	14 … 12	11	10 … 8
Rsvd	Pattern Horizontal Seed	Tiling Enable	Transparency Range Mode

7 … 5	4	3	2	1	0
PatternVertical Seed	DST RMW	Color Source	Mono Source	Color Pattern	Mono Pattern

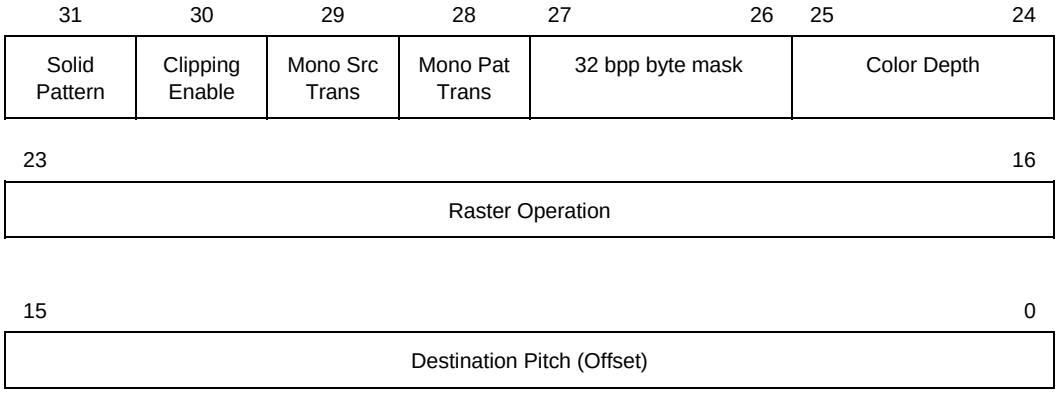
Bit	Descriptions
31	**BLT Engine Busy.** This bit indicates whether the BLT Engine is busy (1) or idle (0). This bit is replicated in the SETUP BLT Opcode & Control register. 1 = Busy 0 = Idle
30	**Setup Instruction Instruction.** The current instruction performs clipping (1).
29	**Setup Monochrome Pattern.** This bit is decoded from the Setup instruction opcode to identify whether a color (0) or monochrome (1) pattern is used with the SCANLINE_BLT instruction. 1 = Monochrome 0 = Color
28:22	**Instruction Target (Opcode).** This is the contents of the Instruction Target field from the last BLT instruction. This field is used by the BLT Engine state machine to identify the BLT instruction it is to perform. The opcode specifies whether the source and pattern operands are color or monochrome.
21:20	**32 bpp byte mask:** 21 = 1 = write alpha channel [31:24]; 20 = 1 = write RGB channels [23:00]. This field is only used for 32bpp.
19:17	**Monochrome Source Start.** This field indicates the starting monochrome pixel bit position within a byte per scan line of the source operand. The monochrome source is word aligned which means that at the end of the scan line all bits should be discarded until the next word boundary.
16	**Bit/Byte Packed.** Byte packed is for the NT driver 0 = Bit 1 = Byte
15	**Src Tiling Enable:** 0 = Tiling Disabled (Linear) 1 = Tiling enabled (Tile-X only)
14:12	**Horizontal Pattern Seed.** This field indicates the pattern pixel position which corresponds to X = 0.
11	**Dest Tiling Enable:** 0 = Tiling Disabled (Linear blit) 1 = Tiling enabled (Tile-X only) When set to '1', this means that Blitter is executing in Tiled-X mode. If '0' it means that Blitter is in Linear mode. Blitter never executes in Tiled-Y mode. On reset, this bit will be '0'. This definition applies to only X,Y Blits. Non-XY blits (COLOR_BLT, SRC_COPY_BLT), will support only linear mode and will not support tiling and for them this bit will remain reserved.

Bit	Descriptions
10:8	**Transparency Range Mode.** These bits control whether or not the byte(s) at the destination corresponding to a given pixel will be conditionally written, and what those conditions are. This feature can make it possible to perform various masking functions in order to selectively write or preserve graphics data already at the destination. XX0 = No color transparency mode enabled. This causes normal operation with regard to writing data to the destination. 001 = **[Source color transparency]** The **Transparency Color Low:** (Pixel Greater or Equal) (source background register) and the **Transparency Color High:** (Pixel Less or Equal) (source foreground register) are compared to the source pixels. The range comparisons are done on each component (R,G,B) and then logically ANDed. If the source pixel components are not within the range defined by the Transparency Color registers, then the byte(s) at the destination corresponding to the current pixel are written with the result of the bit-wise operation. 011 = **[Source and Alpha color transparency]** The **Transparency Color Low:** (Pixel Greater or Equal) (source background register) and the **Transparency Color High:** (Pixel Less or Equal) (source foreground register) are compared to the source pixels. The range comparisons are done on each component (A,R,G,B) and then logically ANDed. If the source pixel components are not within the range defined by the Transparency Color registers, then the byte(s) at the destination corresponding to the current pixel are written with the result of the bit-wise operation. 101 = **[Destination and Alpha color transparency]** The **Transparency Color Low:** (Pixel Greater or Equal) (source background register) and the **Transparency Color High:** (Pixel Less or Equal) (source foreground register) are compared to the destination pixels. The range comparisons are done on each component (A,R,G,B) and then logically ANDed. If the destination pixels are within the range, then the byte(s) at the destination corresponding to the current pixel are written with the result of the bit-wise operation. 111 = **[Destination color transparency]** The **Transparency Color Low:** (Pixel Greater or Equal) (source background register) and the **Transparency Color High:** (Pixel Less or Equal) (source foreground register) are compared to the destination pixels. The range comparisons are done on each component (R,G,B) and then logically ANDed. If the destination pixels are within the range, then the byte(s) at the destination corresponding to the current pixel are written with the result of the bit-wise operation.
7:5	**Pattern Vertical Seed.** This field specifies the pattern scan line which corresponds to Y=0.
4	**Destination Read Modify Write.** This bit is decoded from the last instruction's opcode field and Destination Transparency Mode to identify whether a Destination read is needed.
3	**Color Source.** This bit is decoded from the last instructions opcode field to identify whether a color (1) source is used.
2	**Monochrome Source.** This bit is decoded from the last instructions opcode field to identify whether a monochrome (1) source is used.
1	**Color Pattern.** This bit is decoded from the last instructions opcode field to identify whether a color (1) pattern is used.
0	**Monochrome Pattern.** This bit is decoded from the last instructions opcode field to identify whether a monochrome (1) pattern is used.

14.10.2 BR01—Setup BLT Raster OP, Control, and Destination Offset

Memory Offset Address: none
Default: 0000 xxxx
Attributes: State accessible

BR01 contains the contents of the last Setup instruction DWord 1. It is identical to the BLT Raster OP, Control, and Destination Offset definition, but it is used with the following instructions: PIXEL_BLT, SCANLINE_BLT, and TEXT_BLT.



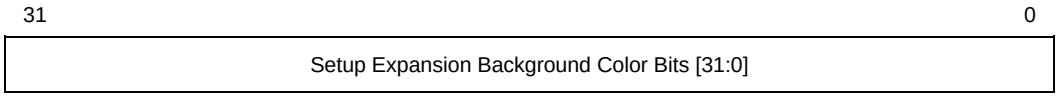
Bit	Descriptions
31	**Solid Pattern Select.** This bit applies only when the pattern data is monochrome. This bit determines whether or not the BLT Engine actually performs read operations from the frame buffer in order to load the pattern data. Use of this feature to prevent these read operations can increase BLT Engine performance, if use of the pattern data is indeed not necessary. The BLT Engine is configured to accept either monochrome or color pattern data via the opcode field. 0 = This causes normal operation with regard to the use of the pattern data. The BLT Engine proceeds with the process of reading the pattern data, and the pattern data is used as the pattern operand for all bit-wise operations. 1 = The BLT Engine forgoes the process of reading the pattern data, the presumption is made that all of the bits of the pattern data are set to 0, and the pattern operand for all bit-wise operations is forced to the background color specified in the Color Expansion Background Color Register.
30	**Clipping Enabled**: 1 = Enabled; 0 = Disabled

Bit	Descriptions
29	**Monochrome Source Transparency Mode.** This bit applies only when the source data is in monochrome. This bit determines whether or not the byte(s) at the destination corresponding to the pixel to which a given bit of the source data also corresponds will actually be written if that source data bit has the value of 0. This feature can make it possible to use the source as a transparency mask. The BLT Engine is configured to accepted either monochrome or color source data via the opcode field. 0 = This causes normal operation with regard to the use of the source data. Wherever a bit in the source data has the value of 0, the color specified in the background color register is used as the source operand in the bit-wise operation for the pixel corresponding to the source data bit, and the bytes at the destination corresponding to that pixel are written with the result. 1 = Wherever a bit in the source data has the value of 0, the byte(s) at the destination corresponding to the pixel to which the source data bit also corresponds are simply not written, and the data at those byte(s) at the destination are allowed to remain unchanged.
28	**Monochrome Pattern Transparency Mode.** This bit applies only when the pattern data is monochrome. This bit determines whether or not the byte(s) at the destination corresponding to the pixel to which a given bit of the pattern data also corresponds will actually be written if that pattern data bit has the value of 1. This feature can make it possible to use the pattern as a transparency mask. The BLT Engine is configured to accepted either monochrome or color pattern data via the opcode field. 0 = This causes normal operation with regard to the use of the pattern data. Wherever a bit in the pattern data has the value of 0, the color specified in the background color register is used as the pattern operand in the bit-wise operation for the pixel corresponding to the pattern data bit, and the bytes at the destination corresponding to that pixel are written with the result. 1 = Wherever a bit in the pattern data has the value of 0, the byte(s) at the destination corresponding to the pixel to which the pattern data bit also corresponds are simply not written, and the data at those byte(s) at the destination are allowed to remain unchanged.
27:26	**32 bpp byte mask.** 21 = 1 = write alpha channel [31:24]; 20 = 1 = write RGB channels [23:00]. This field is only used for 32bpp.
25:24	**Color Depth.** 00 = 8 Bit Color Depth 01 = 16 Bit Color Depth 10 = 16 Bit Color Depth 11 = 32 Bit Color Depth
23:16	**Raster Operation Select.** These 8 bits are used to select which one of 256 possible raster operations is to be performed by the BLT Engine. The opcode field must indicate a monochrome source if ROP = F0.

Bit	Descriptions
15:0	**Destination Pitch (Offset).** For non-XY Blits, the signed 16bit field allows for specifying upto ± 32Kbytes signed pitches in bytes (same as before). For X, Y Blits with tiled (X) surfaces, the pitch for Destination will be 512Byte aligned and should be programmable upto ± 128Kbytes. In this case, this 16bit signed pitch field is used to specify upto ± 32K**DWords**. For X, Y blits with nontiled surfaces (linear surfaces), this 16bit field can be programmed to byte specification of upto ± 32Kbytes (same as before). These 16 bits store the signed memory address offset value by which the destination address originally specified in the Destination Address Register is incremented or decremented as each scan line's worth of destination data is written into the frame buffer by the BLT Engine, so that the destination address will point to the next memory address to which the next scan line's worth of destination data is to be written. If the intended destination of a BLT operation is within on-screen frame buffer memory, this offset is normally set so that each subsequent scan line's worth of destination data lines up vertically with the destination data in the scan line, above. However, if the intended destination of a BLT operation is within off-screen memory, this offset can be set so that each subsequent scan line's worth of destination data is stored at a location immediately after the location where the destination data for the last scan line ended, in order to create a single contiguous block of bytes of destination data at the destination.

14.10.3 BR05—Setup Expansion Background Color

Memory Offset Address: none
Default: None
Attributes: State accessible



Bit	Descriptions
31:0	**Setup Expansion Background Color Bits [31:0].** These bits provide the one, two, or four bytes worth of color data that select the background color to be used in the color expansion of monochrome pattern or source data for either the SCANLINE_BLT or TEXT_BLT instructions. BR05 is also used as the solid pattern for the PIXEL_BLT instruction. Whether one, two, or three bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

14.10.4 BR06—Setup Expansion Foreground Color

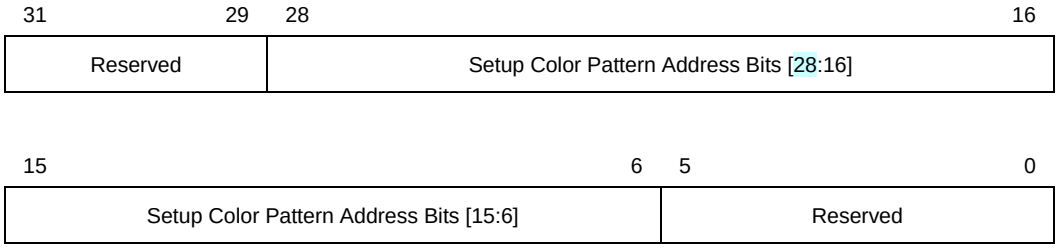
Memory Offset Address: none
Default: None
Attributes: State accessible

31	0
Setup Expansion Foreground Color Bits [31:0]	

Bit	Descriptions
31:24	**Reserved.**
31:0	**Setup Expansion Foreground Color Bits [31:0].** These bits provide the one, two, or four bytes worth of color data that select the foreground color to be used in the color expansion of monochrome pattern or source data for either the SCANLINE_BLT or TEXT_BLT instructions. Whether one, two, or three bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

14.10.5 BR07—Setup Color Pattern Address

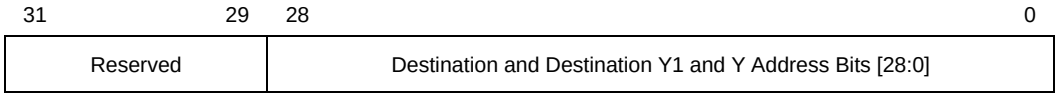
Memory Offset Address: none
Default: None
Attributes: State accessible



Bit	Descriptions
31:29	**Reserved.** The maximum GC graphics address is 512 MBs.
28:6	**Pattern Address.** These 23 bits specify the starting address of the color pattern from the SETUP_BLT instruction. This register works identically to the Pattern Address register, but this version is only used with the SCANLINE_BLT instruction execution. The pattern data must be located in linear memory. The pattern data must be located on a pattern-size boundary. The pattern is always of 8x8 pixels, and therefore, its size is dependent upon its pixel depth. The pixel depth may be 8, 16, or 32 bits per pixel if the pattern is in color (the pixel depth of a color pattern must match the pixel depth to which the graphics system has been set). Monochrome patterns require 8 bytes and are supplied through the instruction. Color patterns of 8, 16, and 32 bits per pixel color depth must start on 64-byte, 128-byte and 256-byte boundaries, respectively.
5:0	**Reserved.** These bits always return 0 when read.

14.10.6 BR09—Destination Address

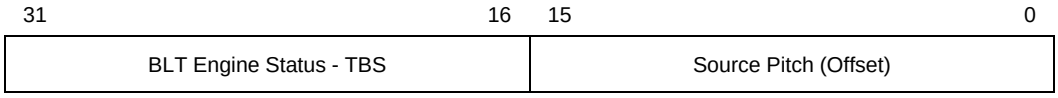
Memory Offset Address: None
Default: None
Attributes: State accessible



Bit	Descriptions
31:29	**Reserved.**
28:0	**Destination Address Bits.** When tiling is enabled for XY-blits, this base address should be limited to 4KB. Otherwise for XY blits, there is no restriction and it is same as before. These 29 bits specify the starting pixel address of the destination data. This register is also the working destination address register and changes as the BLT Engine performs the accesses. Used as the scan line address (Destination Y Address & Destination Y1 Address) for BLT instructions: PIXEL_BLT, SCANLINE_BLT, and TEXT_BLT. In this case the address points to the first pixel in a scan line and is compared with the ClipRect Y1 & Y2 address registers to determine whether the scan line should be written or not. The Destination Y1 address is the top scan line to be written for text. Note that for non-XY blits (COLOR_BLT, SRC_COPY_BLT), this address points to the first byte to be written. This register is always the last register written for a BLT drawing instruction. Writing BR09 starts the BLT engine execution. ***Note:*** Some instructions affect only one scan line (requiring only one coordinate); other instructions affect multiple scan lines and need both coordinates.

14.10.7 BR11—BLT Source Pitch (Offset)

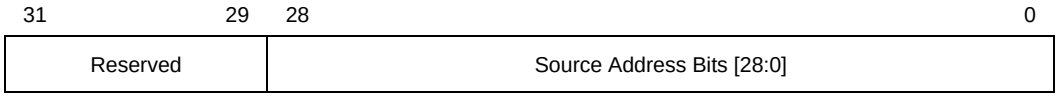
Memory Offset Address: None
Default: None
Attributes: Not accessible



Bit	Descriptions
31:16	**BLT Engine Status.** This field is used to read back important debug status. It will be specified in the future.
15:0	**Source Pitch (Offset)** For non-XY Blits with color source operand (SRC_COPY_BLT), the signed 16bit field allows for specifying upto ± 32Kbytes signed pitch in bytes (same as before). For X, Y Blits with tiled (X) surfaces, the pitch for Color Source will be 512Byte aligned and should be programmable upto ± 128Kbytes. In this case, this 16bit signed pitch field is used to specify upto ± 32K**DWords**. For X, Y blits with nontiled color source surfaces (linear surfaces), this 16bit field can be programmed to byte specification of upto ± 32Kbytes (same as before). When the color source data is located within the frame buffer or AGP aperture, these signed 16 bits store the memory address offset (pitch) value by which the source address originally specified in the Source Address Register is incremented or decremented as each scan line's worth of source data is read from the frame buffer by the BLT Engine, so that the source address will point to the next memory address from which the next scan line's worth of source data is to be read. Note that if the intended source of a BLT operation is within on-screen frame buffer memory, this offset is normally set to accommodate the fact that each subsequent scan line's worth of source data lines up vertically with the source data in the scan line, above. However, if the intended source of a BLT operation is within off-screen memory, this offset can be set to accommodate a situation in which the source data exists as a single contiguous block of bytes where in each subsequent scan line's worth of source data is stored at a location immediately after the location where the source data for the last scan line ended.

14.10.8 BR12—Source Address

Memory Offset Address: None
Default: None
Attributes: Not accessible



Bit	Descriptions
31:29	**Reserved.** The maximum GC Graphics address is 512 MBs.
28:0	**Source Address Bits [28:0].** When tiling is enabled for XY-blits with Color source surfaces, this base address should be limited to 4KB. Otherwise for XY blits, there is no restriction and it is same as before, including for monosource and text blits. Note that for non-XY blit with Color Source (SRC_COPY_BLT), this address points to the first byte to be read. These 29 bits are used to specify the starting pixel address of the color source data. The lower 3 bits are used to indicate the position of the first valid byte within the first Quadword of the source data.

14.10.9 BR13—BLT Raster OP, Control, and Destination Pitch

Memory Offset Address: None
Default: 0000 xxxx
Attributes: Not accessible

Bit	Descriptions
31	**Solid Pattern Select.** This bit applies only when the pattern data is monochrome. This bit determines whether or not the BLT Engine actually performs read operations from the frame buffer in order to load the pattern data. Use of this feature to prevent these read operations can increase BLT Engine performance, if use of the pattern data is indeed not necessary. The BLT Engine is configured to accept either monochrome or color pattern data via the opcode field. 0 = This causes normal operation with regard to the use of the pattern data. The BLT Engine proceeds with the process of reading the pattern data, and the pattern data is used as the pattern operand for all bit-wise operations. 1 = The BLT Engine forgoes the process of reading the pattern data, the presumption is made that all of the bits of the pattern data are set to 0, and the pattern operand for all bit-wise operations is forced to the background color specified in the Color Expansion Background Color Register.
30	**Clipping Enabled**: 1 = Enabled; 0 = Disabled
29	**Monochrome Source Transparency Mode.** This bit applies only when the source data is in monochrome. This bit determines whether or not the byte(s) at the destination corresponding to the pixel to which a given bit of the source data also corresponds will actually be written if that source data bit has the value of 0. This feature can make it possible to use the source as a transparency mask. The BLT Engine is configured to accepted either monochrome or color source data via the opcode field. 0 = This causes normal operation with regard to the use of the source data. Wherever a bit in the source data has the value of 0, the color specified in the background color register is used as the source operand in the bit-wise operation for the pixel corresponding to the source data bit, and the bytes at the destination corresponding to that pixel are written with the result. 1 = Where a bit in the source data has the value of 0, the byte(s) at the destination corresponding to the pixel to which the source data bit also corresponds are simply not written, and the data at those byte(s) at the destination are allowed to remain unchanged.
28	**Monochrome Pattern Transparency Mode.** This bit applies only when the pattern data is monochrome. This bit determines whether or not the byte(s) at the destination corresponding to the pixel to which a given bit of the pattern data also corresponds will actually be written if that pattern data bit has the value of 1. This feature can make it possible to use the pattern as a transparency mask. The BLT Engine is configured to accepted either monochrome or color pattern data via the opcode in the Opcode and Control register. 0 = This causes normal operation with regard to the use of the pattern data. Where a bit in the pattern data has the value of 0, the color specified in the background color register is used as the pattern operand in the bit-wise operation for the pixel corresponding to the pattern data bit, and the bytes at the destination corresponding to that pixel are written with the result. 1= Wherever a bit in the pattern data has the value of 0, the byte(s) at the destination corresponding to the pixel to which the pattern data bit also corresponds are simply not written, and the data at those byte(s) at the destination are allowed to remain unchanged.
25:24	**Color Depth.** 00 = 8 Bit Color Depth 01 = 16 Bit Color Depth 10 = 24 Bit Color Depth 11 = Reserved
23:16	**Raster Operation Select.** These 8 bits are used to select which one of 256 possible raster operations is to be performed by the BLT Engine. The opcode must indicate a monochrome source operand if ROP = F0.

Bit	Descriptions
15:0	**Destination Pitch (Offset).** These 16 bits store the signed memory address offset value by which the destination address originally specified in the Destination Address Register is incremented or decremented as each scan line's worth of destination data is written into the frame buffer by the BLT Engine, so that the destination address will point to the next memory address to which the next scan line's worth of destination data is to be written. If the intended destination of a BLT operation is within on-screen frame buffer memory, this offset is normally set so that each subsequent scan line's worth of destination data lines up vertically with the destination data in the scan line, above. However, if the intended destination of a BLT operation is within off-screen memory, this offset can be set so that each subsequent scan line's worth of destination data is stored at a location immediately after the location where the destination data for the last scan line ended, in order to create a single contiguous block of bytes of destination data at the destination.

14.10.10 BR14—Destination Width & Height

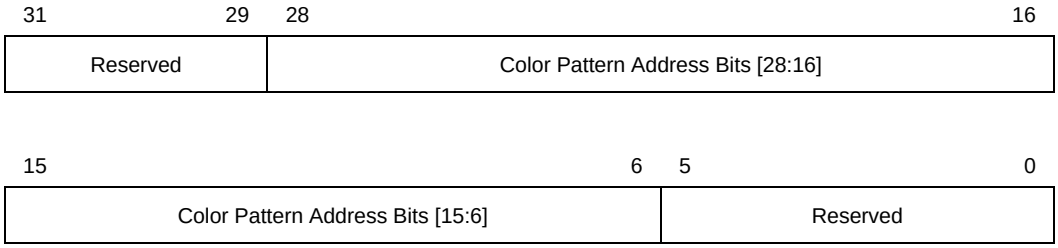
Memory Offset Address: None
Default: None
Attributes: Not accessible

BR14 contains the values for the height and width of the data to be BLT. If these values are not correct, such that the BLT Engine is either expecting data it does not receive or receives data it did not expect, the system can hang.

Bit	Descriptions
31:29	**Reserved.**
28:16	**Destination Height.** These 13 bits specify the height of the destination data in terms of the number of scan lines. This is a working register.
15:13	**Reserved.**
12:0	**Destination Byte Width.** These 13 bits specify the width of the destination data in terms of the number of bytes per scan line. The number of pixels per scan line into which this value translates depends upon the color depth to which the graphics system has been set.

14.10.11 BR15—Color Pattern Address

Memory Offset Address: None
Default: None
Attributes: Not accessible

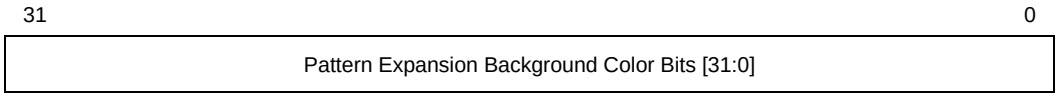


.

Bit	Descriptions
31:29	**Reserved.** The maximum GC graphics address is 512 MBs.
28:6	**Color Pattern Address.** There is no change to the Color Pattern address specification due to Non-Power-of-2 change. It remains the same as before. The pattern data must be located in linear memory. These 23 bits specify the starting address of the pattern. The pattern data must be located on a pattern-size boundary. The pattern is always of 8x8 pixels, and therefore, its size is dependent upon its pixel depth. The pixel depth may be 8, 16, or 32 bits per pixel if the pattern is in color (the pixel depth of a color pattern must match the pixel depth to which the graphics system has been set). Monochrome patterns require 8 bytes and are applied through the instruction. Color patterns of 8, 16, and 32 bits per pixel color depth must start on 64-byte, 128-byte and 256-byte boundaries, respectively.
5:0	**Reserved.** These bits always return 0 when read.

14.10.12 BR16—Pattern Expansion Background & Solid Pattern Color

Memory Offset Address: 40040h
Default: None
Attributes: RO; DWord accessible

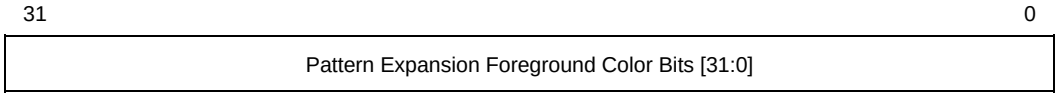


.

Bit	Descriptions
31:0	**Pattern Expansion Background Color Bits [31:0].** These bits provide the one, two, or four bytes worth of color data that select the background color to be used in the color expansion of monochrome pattern data during BLT operations. Whether one, two, or four bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

14.10.13 BR17—Pattern Expansion Foreground Color

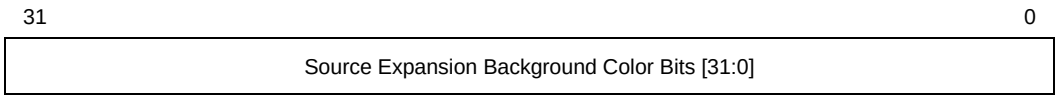
Memory Offset Address: None
Default: None
Attributes: Not accessible



Bit	Descriptions
31:0	**Pattern Expansion Foreground Color Bits [31:0].** These bits provide the one, two, or four bytes worth of color data that select the foreground color to be used in the color expansion of monochrome pattern data during BLT operations. Whether one, two, or four bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

14.10.14 BR18—Source Expansion Background, and Destination Color

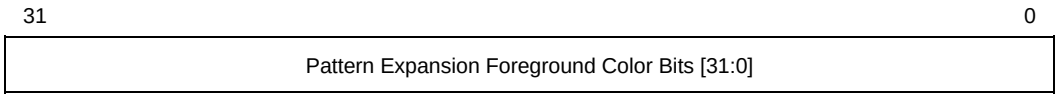
Memory Offset Address: None
Default: None
Attributes: Not accessible



Bit	Descriptions
31:0	**Source Expansion Background Color Bits [31:0].** These bits provide the one, two, or four bytes worth of color data that select the background color to be used in the color expansion of monochrome source data during BLT operations. This register is also used to support destination transparency mode and Solid color fill. Whether one, two, three, or four bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

14.10.15 BR19—Source Expansion Foreground Color

Memory Offset Address: None
Default: None
Attributes: Not accessible



Bit	Descriptions
31:0	**Pattern/Source Expansion Foreground Color Bits [31:0].** These bits provide the one, two, or four bytes worth of color data that select the foreground color to be used in the color expansion of monochrome source data during BLT operations. Whether one, two, or four bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.