Intel® HD Graphics OpenSource PRM

Volume 1 Part 4: Graphics Core – Video Codec Engine

For the all new 2010 Intel Core Processor Family
Programmer's Reference Manual (PRM)

February 2010

Revision 1.0

IHD-OS-022810-R1V1PT4

Contents

1. Video Codec Engine Command Streamer....5
1.1 Registers for Video Codec 5
1.1.1 Introduction....5
1.1.2 Virtual Memory Control....5
1.1.3 Mode and Misc Ctrl Registers 9
1.1.4 Context Submission....13
1.1.5 VCS_RINGBUF—Ring Buffer Registers....15
1.1.6 Watchdog Timer Registers....19
1.1.7 Interrupt Control Registers 20
1.1.8 Logical Context Support 28
1.2 Memory Interface Commands for Video Codec Engine....32
1.2.1 Introduction....32
1.2.2 MI_ARB_CHECK....32
1.2.3 MI_BATCH_BUFFER_START 33
1.2.4 MI_LOAD_REGISTER_IMM 35
1.2.5 MI_NOOP 36
1.2.6 MI_REPORT_HEAD....37
1.2.7 MI_STORE_DATA_IMM....38
1.2.8 MI_STORE_DATA_INDEX....39
1.2.9 MI_SUSPEND_FLUSH 40
1.2.10 MI_USER_INTERRUPT....41
1.2.11 MI_WAIT_FOR_EVENT....41

Revision History

Document Number	Revision Number	Description	Revision Date
IHD-OS-022810-R1V1PT4	1.0	First Release.	February 2010

§§

1. Video Codec Engine Command Streamer

VCE has its own command streamer and operates completely independently of the render (3D/Media) pipeline command streamer.

1.1 Registers for Video Codec

1.1.1 Introduction

This command streamer supports a completely independent set of registers. Only a subset of the MI Registers is supported for this 2nd command streamer. The effort is to keep the registers at the same offset as the render command streamer registers. The base of the registers for the video decode engine will be defined per project, the offsets will be maintained.

1.1.2 Virtual Memory Control

MFX engine Supports a 2-level mapping scheme for PPGTT, consisting of a first-level page directory containing page table base addresses, and the page tables themselves on the 2nd level, consisting of page addresses.

1.1.2.1 VCS_PP_DIR_BASE – Page Directory Base Register

VCS_PP_DIR_BASE – Page Directory Base Register	
Register Type:	MMIO_VCS
Address Offset:	12390h
Project:	All
Default Value:	00000000h
Access:	R/W
Size (in bits):	32
Trusted Type:	1

This register contains the offset into the GGTT where the (current context's) PPGTT page directory begins. This register is restored with context

Bit	Description
31:16	**Page Directory Base Offset** Project: All Security: None Default Value: 0h DefaultVaueDesc Format: U15 Address: GraphicsAddress[31:16] Range [0,PPGTT Size - 1 in cachelines] Contains the cacheline (64-byte) address into the GGTT where the page directory begins.
15:0	**Reserved** Project: All Format: MBZ

1.1.2.2 VCS_PP_DCLV – PPGTT Directory Cacheline Valid Register

VCS_PP_DCLV – PPGTT Directory Cacheline Valid Register

Register Type:	MMIO_CS
Address Offset:	12220h
Project:	All
Default Value:	0h
Access:	R/W
Size (in bits):	64

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

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

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

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

This field below needs to go in some register to enable PPGTT (please review and change description if necessary). Either in GAC MMIO or VCS MMIO

1	**Per-Process GTT Enable** Project: DevGT+ Format: Enable If set, PPGTT support in hardware is enabled. Setting this bit also allows support for big pages (32k)

1.1.2.3 VCS_HWS_PGA — Hardware Status Page Address Register

Address Offset: 14080h–14083h
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.

1.1.3 Mode and Misc Ctrl Registers

1.1.3.1 VCS_MI_MODE — Mode Register for Software Interface

Address Offset: 1209Ch–1209Fh
Default Value: 0000 0000h
Access: Read/Write
Size: 32 bits

The MI_MODE register contains information that controls software interface aspects of the command parser.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:16</td><td>Masks: A “1” in a bit in this field allows the modification of the corresponding bit in Bits 15:0</td></tr>
<tr><td>15</td><td>Suspend Flush
Project: All
Mask: MMIO(0x209c)#31
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>0h</td><td>No Delay</td><td>HW will not delay flush, this bit will get cleared by MI_SUSPEND_FLUSH as well</td><td>All</td></tr>
<tr><td>1h</td><td>Delay Flush</td><td>HW will delay the flush because of sync flush or VTD regimes until reset, this bit will get set by MI_SUSPEND_FLUSH as well</td><td>All</td></tr>
</table></td></tr>
<tr><td>14:12</td><td>Reserved Read/Write</td></tr>
<tr><td>11</td><td>Invalidate UHPTR enable: If bit set H/W clears the valid bit of BCS_UHPTR (4134h, bit 0) when current active head pointer is equal to UHPTR.</td></tr>
<tr><td>10</td><td>Reserved Read/Write</td></tr>
<tr><td>9</td><td>Ring Idle (Read Only Status bit)
0 = Parser not Idle
1 = Parser Idle
Writes to this bit are not allowed.</td></tr>
</table>

Bit	Description
8	**Stop Ring** 0 = Normal Operation. 1 = Parser is turned off. Software must set this bit to force the Ring and Command Parser to Idle. Software must read a “1” in Ring Idle bit after setting this bit to ensure that the hardware is idle. ***Software must clear this bit for Ring to resume normal operation.***
7:2	**Reserved** Read/Write

1.1.3.2 VCS_INSTPM—Instruction Parser Mode Register

Address Offset: 120C0h–120C3h
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.
15:7	Reserved: MBZ
6	**Memory Sync Enable:** This set, this bit allows the video decode engine to write out the data from the local caches to memory.
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*). **<u>Programming Note:</u>** • 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

1.1.3.3 VCS_NOPID — NOP Identification Register

Address Offset:	12094h–12097h
Default Value:	0000 0000h
Access:	Read Only
Size:	32 bits

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

Bit	Description
31:22	Reserved: MBZ
21:0	**Identification Number:** This field contains the 22-bit Noop Identification value specified by the last MI_NOOP instruction that enabled this field to be updated.

1.1.3.4 VCS_EXCC—Execute Condition Code Register

VCS_EXCC—Execute Condition Code Register

Register Type:	MMIO_VCS
Address Offset:	12028h
Project:	All
Default Value:	00000000h
Access:	R/W,RO
Size (in bits):	32
Trusted Type:	1

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

Bit	Description
31:16	**Mask Bits** Format: Mask[1] This bit serves as a write enable for bit 1. If this register is written with this bit clear the corresponding bit in the field 1 will not be modified. Reading these bits always returns 0s.
15:2	**Reserved** Project: All Format: MBZ
4:0	**User Defined Condition Codes** The software may signal a Stream Semaphore by setting the Mask bit and Signal Bit together to match the bit field specified in a WAIT_FOR_EVENT (Semaphore).

1.1.3.5 VBSYNC – Video/Blitter Semaphore Sync Register

VBSYNC – Vidoe/Blitter Semaphore Sync Register	
Register Type: MMIO_VCS **Address Offset:** 12040h **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 32 **Trusted Type:** 1	
This register is written by BCS, read by VCS.	
Bit	**Description**
31:0	**Semaphore Data** Semaphore data for synchronization between video codec engine and blitter engine..

1.1.3.6 VRSYNC – Video/Render Semaphore Sync Register

VRSYNC – Video/Render Semaphore Sync Register	
Register Type: MMIO_VCS **Address Offset:** 12044h **Project:** All **Default Value:** 00000000h **Access:** R/W **Size (in bits):** 32 **Trusted Type:** 1	
This register is written by CS, read by VCS.	
Bit	**Description**
31:0	**Semaphore Data** Semaphore data for synchronization between video codec engine and render engine.

1.1.4 Context Submission

1.1.4.1 VCS_RCCID—Ring Buffer Current Context ID Register

Address Offset: 127C0h–127C4h
Default Value: 00 00 00 00h
Access: Read/Write
Size: 32 bits

This register contains the current "ring context ID" associated with the ring buffer.

Programming Notes:

- The current context registers must not be written directly (via MMIO). The RCCID register should only be updated indirectly from RNCID.

Bit	Description
63:0	See Context Descriptor for VCS

1.1.4.2 VCS_RNCID—Ring Buffer Next Context ID Register

Address Offset: 12700h–12708h
Default Value: 00 00 00 00h
Access: Read/Write
Size: 64 bits

This register contains the *next* "ring context ID" associated with the ring buffer.

Programming Notes:

- The current context (RCCID) register can be updated indirectly from this register on a context switch event. Note that this can only be triggered when arbitration is enabled or if the current context runs dry (head pointer becomes equal to tail pointer).

Bit	Description
63:0	See Context Descriptor for VCS

1.1.4.3 Context Status

A context switch interrupt will be sent anytime a context switch change occurs. This is documented in the "GPU Overview" volume, "Memory Data Formats" chapter. A status DW for the context that was just switched away from will be written to the Context Status Buffer in the Global Hardware Status Page. The status contains the context ID and the reason for the context switch. Note that since there were no running contexts when the very first (after reset) context is submitted, the Context ID in the first Context Status DWord will be UNDEFINED.

Table 1-1. Format of Context Status Dword

Bit	Description
31:12	**Context ID.** Contains the context ID copied from the submitted context.
11:8	Reserved: MBZ
7	**Media watch dog timer expired** cause the context switch
6	Reserved: MBZ
5	Reserved: MBZ
4	**Ring Buffer Becoming Empty** Caused context to Switch.
3	Reserved: MBZ
2	Reserved: MBZ
1	**Waiting on a Semaphore** Caused Context to Switch.
0	Reserved: MBZ

When SW services a context switch interrupt, it should read the Context Status Buffer beginning where it left off reading the last time it serviced a context switch interrupt. It should read up through the **Last Written Status Offset**, which is also recorded in the Context Status Buffer. The status DWs can be examined to determine which contexts were switched out between context interrupt service intervals, and why.

Table 1-2. Number of Context Status Entries in Memory

Device	Number of Status Entries
DevSNB	12 (DW) Entries

Status Dwords are written out to the Context Status Buffer at incrementing addresses. The Context Status Buffer has a limited size (see **Error! Reference source not found.**) and simply wraps around to the beginning when the end is reached. The Context Status Buffer fits into a single cacheline so that the whole buffer will be read from memory at once if the driver performs a cacheable read.

Table 1-3. Format of the Context Status Buffer

DW	Description
15	**Last Written Status Offset.** This Dword is written on every context switch with the (pre-increment) value of the **Context Status Buffer Pointer Register**. The lower 4 bits increment for every status Dword write; the upper 28 bits are always 0. The lowest 4 bits indicate which of the Context Status Dwords was just written.
14-12	Reserved: MBZ
11-0	**Context Status Dwords.** A circular buffer of context status DWs. As each context is switched away from, its status is written here at ascending DWs as indicated by the **Last Written Status Offset**. Once DW 11 has been written, the pointer wraps around so that the next status will be written at DW0. Format = ContextStatusDW

1.1.5 VCS_RINGBUF—Ring Buffer Registers

Address Offset:	12030h – 0403Fh: Ring Buffer: offset 0h = _TAIL offset 4h = _HEAD offset 8h = _START offset Ch = _CTL
Default Value:	0000 0000h
Access:	Read/32 bit Write Only
Size:	4 DWords / Ring Buffer

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

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

The format of the Ring Buffer register set follows:

DWord Offset	Bit	Description
0	31:21	Reserved: MBZ
	20:3	**Tail Offset:** This field is written by software to specify where the valid instructions placed in the ring buffer end. The value written points to the QWord *past* the last valid QWord of instructions. In other words, it can be defined as the *next* QWord that software will write instructions into. Software must write subsequent instructions to QWords following the Tail Offset, possibly wrapping around to the top of the buffer (i.e., software can't skip around within the buffer). Note that all DWords prior to the location indicated by the **Tail Offset** must contain valid instruction data – which may require instruction padding by software. See **Head Offset** for more information. Format = U18 QWord Offset
	2:0	Reserved: MBZ
1	31:21	**Wrap Count:** This field is incremented by 1 whenever the **Head Offset** wraps from the end of the buffer back to the start (i.e., whenever it wraps back to 0). Appending this field to the **Head Offset** field effectively creates a virtual 4GB Head "Pointer" which can be used as a tag associated with instructions placed in a ring buffer. The Wrap Count itself will wrap to 0 upon overflow. The Wrap Count will get cleared as a result of writes of the Starting Address field. Format = U11 count of ring buffer wraps
	20:2	**Head Offset:** This field indicates the offset of the *next* instruction DWord to be parsed. Software will initialize this field to select the first DWord to be parsed once the RB is enabled. (Writing the Head Offset while the RB is enabled is UNDEFINED). Subsequently, the device will increment this offset as it executes instructions – until it reaches the QWord specified by the **Tail Offset**. At this point the ring buffer is considered "empty". **Programming Notes:** • A RB can be enabled empty or containing some number of valid instructions. • Head Offset is cleared as a result of writes of the Starting Address field. Format = U19 DWord Offset
	1:0	Reserved: MBZ

DWord Offset	Bit	Description
2	31:12	**Starting Address:** This field specifies Bits 31:12 of the 4KB-aligned starting Graphics Address of the ring buffer. 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**. Format: Graphics Address Bits 31:12
	11:0	Reserved: MBZ
3	31:21	Reserved: MBZ
	20:12	**Buffer Length:** This field is written by SW to specify the length of the ring buffer in 4 KB Pages. Format = U9 in 4 KB pages – 1 Range = [0 = 1 page = 4 KB, 1FFh = 512 pages = 2 MB]
	11	**RBWait** Indicates that this ring has executed a WAIT_FOR_EVENT instruction and is currently waiting. Software can write a "1" to clear this bit, write of "0" has no effect. When the RB is waiting for an event and this bit is cleared, the wait will be terminated and the RB will be returned to arbitration.
	10	**Semaphore Wait** Indicates that this ring has executed a MI_SEMAPHORE_MBOX instruction with register compare and is currently waiting. Software can write a "1" to clear this bit, write of "0" has no effect. When the RB is waiting for the compare to meet and this bit is cleared, the wait will be terminated and the RB will be returned to arbitration.
	9	**Reserved:** MBZ
	8	**Disable Register Accesses:** 0 = Ring is allowed to access (read or write) MMIO space. 1 = Ring is not allowed to write MMIO space. Ring *is* allowed to read registers.
	7:3	Reserved: MBZ

DWord Offset	Bit	Description
	2:1	**Automatic Report Head Pointer:** This field is written by software to control the automatic “reporting” (write) of this ring buffer’s “Head Pointer” register (register DWord 1) to the corresponding location within the Hardware Status Page. Automatic reporting can either be disabled or enabled at 4KB, 64KB or 128KB boundaries within the ring buffer. Format = 0: MI_AUTOREPORT_OFF – Automatic reporting disabled 1: MI_AUTOREPORT_64KB – Report every 16 pages (64KB) 2: MI_AUTOREPORT_4KB – Report every page (4KB) 3: MI_AUTOREPORT_128KB – Report every 32 pages (128KB) When the **Per-Process Virtual Address Space Enable** bit is set and automatic head reporting is desired, this field must be set to option 2 since the ring buffer will be only 16KB in size. The head pointer will be reported to the head pointer location in the PP HW Status Page when it passes each 4KB page boundary. When the above-mentioned bit is set, reporting will behave just as on the prior devices (as documented above), and option 2 is not legal.
	0	**Ring Buffer Enable:** This field is used to enable or disable this ring buffer. It can be enabled or disabled regardless of whether there are valid instructions pending. Format = Enable

1.1.5.1 VCS_UHPTR — Pending Head Pointer Register

Address Offset: 12134h–12137h
Default Value: 0000 0000h
Access: Read/Write
Size: 32 bits

Bit	Description
31:3	**Head Pointer Address**: This register represents the GFX address offset where execution should continue in the ring buffer following execution of an MI_ARB_CHECK command. Format = MI_Graphics_Offset
2:1	Reserved: MBZ
0	**Head Pointer Valid:** 1 = Indicates that there is an updated head pointer programmed in this register 0 = No valid updated head pointer register, resume execution at the current location in the ring buffer This bit is set by the software to request a pre-emption. It is reset by hardware 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.

1.1.6 Watchdog Timer Registers

1.1.6.1 VCS_CNTR—Counter for the bit stream decode engine

Address Offset: 12178h–1217Bh
Default Value: FFFF FFFFh
Access: Read/Write
Size: 32 bits

Bit	Description
31:0	**Count Value**: Writing a Zero value to this register starts the counting. Writing a Value of FFFF FFFF to this counter stops the counter

1.1.6.2 VCS_THRSH—Threshold for the counter of bit stream decode engine

Address Offset: 1217Ch–1217Fh
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.

1.1.7 Interrupt Control Registers

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

Table 1-4. Bit Definition for Interrupt Control Registers

Bit	Description
31:9	**Reserved. MBZ:** These bits may be assigned to interrupts on future products/steppings.
8	**Context Switch Interrupt:** Set when a context switch has just occurred. **Per-Process Virtual Address Space Enable bit** needs to be set for this interrupt to occur**.**
7	**Page Fault:** This bit is set whenever there is a pending PPGTT (page or directory) fault.
6	**Timeout Counter Expired:** Set when the VCS timeout counter has reached the timeout thresh-hold value.
5	**Reserved:** MBZ
4	**MI_FLUSH_DW Notify Interrupt:** The Pipe Control packet (Fences) specified in *3D pipeline* document may optionally generate an Interrupt. The Store QW associated with a fence is completed ahead of the interrupt.
3	**Render Command Parser Master Error:** When this status bit is set, it indicates that the hardware has detected an error. It is set by the device upon an error condition and cleared by a CPU write of a one to the appropriate bit contained in the Error ID register followed by a write of a one to this bit in the IIR. Further information on the source of the error comes from the "Error Status Register" which along with the "Error Mask Register" determine which error conditions will cause the error status bit to be set and the interrupt to occur. **Page Table Error:** Indicates a page table error. **Instruction Parser Error**: The Renderer Instruction Parser encounters an error while parsing an instruction.
2	**Sync Status:** This bit is 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).

Bit	Description
1	**Reserved:** MBZ
0	**Render Command Parser User Interrupt:** This status bit is set when an MI_USER_INTERRUPT instruction is executed on the Render Command Parser. Note that instruction execution is not halted and proceeds normally. A mechanism such as an MI_STORE_DATA instruction is required to associate a particular meaning to a user interrupt.

The following table specifies the settings of interrupt bits stored upon a “Hardware Status Write” due to ISR changes:

Bit	Interrupt Bit	ISR bit Reporting via Hardware Status Write (when unmasked via HWSTAM)
8	**Context Switch Interrupt:** Set when a context switch has just occurred.	Not supported to be unmasked
7	**Page Fault:** This bit is set whenever there is a pending PPGTT (page or directory) fault.	Set when event occurs, cleared when event cleared
6	**Media Decode Pipeline Counter Exceeded Notify Interrupt:** The counter threshold for the execution of the media pipeline is exceeded. Driver needs to attempt hang recovery.	Not supported to be unmasked
5	**Reserved**	
4	MI_FLUSH_DW packet - Notify Enable	0
3	Master Error	Set when error occurs, cleared when error cleared
2	Sync Status	Toggled every SyncFlush Event
0	User Interrupt	0

1.1.7.1 HWSTAM — Hardware Status Mask Register

Hardware Status Mask Register	
Register Type:	MMIO_VCS
Address Offset:	12098h
Project:	All
Default Value:	**FFFF FFFFh**
Access:	R/W
Size (in bits):	32
Trusted Type:	1

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

Bit	Description
31:0	**Hardware Status Mask Register** Project: All Default Value: FFFFFFFFh DefaultVaueDesc Format: Array of Masks refer to **Error! Reference source not found.** in Interrupt Control Register section for bit definitions

1.1.7.2 IMR—Interrupt Mask Register

IMR—Interrupt Mask Register	
Register Type:	MMIO_VCS
Address Offset:	120A8h
Project:	All
Default Value:	FFFF FFFFh
Access:	R/W
Size (in bits):	32

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

Bit	Description
31:0	**Interrupt Mask Bits** Project: All Default Value: FFFF FFFFh Format: Array of interrupt mask bits — Refer to Table 1 4 in Interrupt Control Register section for bit definitions This field contains a bit mask which selects which interrupt bits (from the ISR) are reported in the IIR.

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

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

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

The following table describes the Hardware-Detected Error bits:

Table 1-5. Hardware-Detected Error Bits

Bit	Description
15:5	Reserved: MBZ
4	**Page Table Error**: This bit is set when a Graphics Memory Mapping Error is detected. The cause of the error is indicated (to some extent) in the PGTBL_ER register. Note: This error indications can not be cleared except by reset (i.e., it is a fatal error). 1 = Page table error
1	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: 1: Instruction Error detected Programming Note: [DevBW][DevCL]: The bit for the error mask of this register is reserved. The mask should be set to a value of 1.

1.1.7.3.1 EIR — Error Identity Register

EIR — Error Identity Register	
Register Type: MMIO_VCS **Address Offset:** 120B0h **Project:** All **Default Value:** 0000 0000h **Access:** R/WC **Size (in bits):** 32	
The EIR register contains the persistent values of Hardware-Detected Error Condition bits. Any bit set in this register will cause the Master Error bit in the ISR to be set. The EIR register is also used by software to clear detected errors (by writing a '1' to the appropriate bit(s)).	
Bit	**Description**
31:16	**Reserved** Project: All Format: MBZ
15:0	**Error Identity Bits** Project: All Default Value: 0h Format: Array of Error condition bits — See Table 1 5. Hardware-Detected Error Bits This register contains the persistent values of ESR error status bits that are unmasked via the EMR register. (See **Error! Reference source not found.**). The logical OR of all (defined) bits in this register is reported in the Master Error bit of the Interrupt Status Register. In order to clear an error condition, software must first clear the error by writing a '1' to the appropriate bit(s) in this field. If required, software should then proceed to clear the Master Error bit of the IIR.

Value	Name	Description	Project
1h	Error occurred	Error occurred	All

Programming Notes	Project
Writing a '1' to a set bit will cause that error condition to be cleared. However, the Page Table Error bit (Bit 4) can not be cleared except by reset (i.e., it is a fatal error).	All

1.1.7.3.2 EMR—Error Mask Register

EMR—Error Mask Register	
Register Type: MMIO_VCS **Address Offset:** 120B4h **Project:** All **Default Value:** FFFF FFFFh **Access:** R/W **Size (in bits):** 32	
The EMR register is used by software to control which Error Status Register bits are “masked” or “unmasked”. “Unmasked” bits will be reported in the EIR, thus setting the Master Error ISR bit and possibly triggering a CPU interrupt, and will persist in the EIR until cleared by software. “Masked” bits will not be reported in the EIR and therefore cannot generate Master Error conditions or CPU interrupts.	
Bit	**Description**
31:16	**Reserved** Project: All Format: MBZ
15:0	**Error Mask Bits** Project: All Default Value: FFFF FFFFh Format: Array of error condition mask bits — See Table 1 5. Hardware-Detected Error Bits This register contains a bit mask that selects which error condition bits (from the ESR) are reported in the EIR.

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

1.1.7.3.3 ESR—Error Status Register

ESR—Error Status Register	
Register Type:	MMIO_VCS
Address Offset:	120B8h
Project:	All
Default Value:	0000 0000h
Access:	RO
Size (in bits):	32

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

Bit	Description
31:16	**Reserved** Project: All Format: MBZ
15:0	**Error Status Bits** Project: All Default Value: 0h Format: Array of error condition bits — See Table 1 5. Hardware-Detected Error Bits This register contains the non-persistent values of all hardware-detected error condition bits.

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

1.1.8 Logical Context Support

1.1.8.1 VCS_BB_ADDR—Batch Buffer Head Pointer Register

Address Offset:	012140h–012147h
Default Value:	0000 0000 0000 0000h
Access:	Read-Only
Size:	64 bits

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

Bit	Description
63:32	Reserved: MBZ
31:3	**Batch Buffer Head Pointer:** This field specifies the QWord-aligned Graphics Memory Address where the last initiated Batch Buffer is currently fetching commands. If no batch buffer is currently active, the Valid bit will be 0 and this field will be meaningless. .
2:1	Reserved: MBZ
0	**Valid:** 1 = Batch buffer Valid 0 = Batch buffer Invalid

1.1.8.2 VCS_BB_STATE — Batch Buffer State Register

VCS_BB_STATE – Batch Buffer State Register	
Register Type:	MMIO_VCS
Address Offset:	12110h
Project:	All
Default Value:	0000 0000h
Access:	R/W
Size (in bits):	32

This register contains the attributes of the last batch buffer initiated from the Ring Buffer. These include the 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.
4:0	**Reserved** Project: All Format: MBZ

Value	Name	Description	Project
0h	MIBUFFER_SECURE	Located in GGTT memory	All
1h	MIBUFFER_NONSECURE	Located in PPGTT memory	All

1.1.8.3 VCS_CTXT_SR_CTL — Context Save/Restore Control Register

CTXT_SR_CTL – Context Save/Restore Control Register	
Register Type: MMIO_VCS **Address Offset:** 12114h **Project:** All **Default Value:** 0000 0000h **Access:** R/W **Size (in bits):** 32	
This register is saved and restored with context.	
Bit	**Description**
31:1	**Reserved** Project: All Format: MBZ
0	**MFX Context Restore Inhibit** Project: All Format: U1 This is not a true register bit. This bit should be set in the context image of a ring context that is being submitted for the first time. Setting this bit will inhibit the restoring of render context (including extended context if applicable) so that restoring of an uninitialized render context can be prevented. This bit will always be set on a context save (since the render context cannot be uninitialized on context save – it will always contain at least default values.)

1.1.8.4 MFC_BITSTREAM_SE_BITCOUNT —Bitstream Output Bit Count for the last Syntax Element Register

MFC_BITSTREAM_SE_BITCOUNT	
Register Type: MMIO_VCS **Address Offset:** 1240Ch **Project:** All **Default Value:** 00000000h; 00000000h; **Access:** R/W **Size (in bits):** 32 **Trusted Type:** 1	
This register stores the count of number of bits in the bitstream for the last syntax element before padding. The bit count is before the byte-aligned alignment padding insertion, but includes the stop-one-bit. This register is part of the context save and restore.	
Bit	**Description**
31:0	**MFC Bitstream Syntax Element Bit Count** Total number of bits in the bitstream output before padding. This count is updated each time the internal counter is incremented.

1.1.8.5 MFC_AVC_CABAC_INSERTION_COUNT —Bitstream Output CABAC Insertion Count Register

MFC_AVC_CABAC_INSERTION_COUNT	
Register Type:	MMIO_VCS
Address Offset:	12410h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	32
Trusted Type:	1

This register stores the count in bytes of **CABAC ZERO_WORD insertion. It is primarily provided for statistical data gathering.** This register is part of the context save and restore.

Bit	Description
31:0	**MFC AVC Cabac Insertion Count** Total number of bytes in the bitstream output before for the CABAC zero word insertion. This count is updated each time when the insertion count is incremented.

1.1.8.6 MFC_AVC_MINSIZE_PADDING_COUNT —Bitstream Output Minimal Size Padding Count Register

MFC_AVC_MINSIZE_PADDING_COUNT	
Register Type:	MMIO_VCS
Address Offset:	12414h
Project:	All
Default Value:	00000000h; 00000000h;
Access:	R/W
Size (in bits):	32
Trusted Type:	1

This register stores the count in bytes of **minimal size padding insertion. It is primarily provided for statistical data gathering.** This register is part of the context save and restore.

Bit	Description
31:0	**MFC AVC MinSize Padding Count** Total number of bytes in the bitstream output contributing to minimal size padding operation. This count is updated each time when the padding count is incremented.

1.2 Memory Interface Commands for Video Codec Engine

1.2.1 Introduction

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

This chapter describes MI Commands for the Video Codec Engine. Note that these commands are not applicable to [DevBW] and [DevCL] (these devices do not have a parallel Video Codec Engine).

The commands detailed in this chapter are used across the later 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 details.

1.2.2 MI_ARB_CHECK

The MI_ARB_CHECK instruction is used with the UHPTR register. This instruction can be used to pre-empt the current execution of the ring buffer. Note that the valid bit in the UHPTR register needs to be set for the command streamer to be pre-empted.

Programming Note:

- This instruction can be placed only in a ring buffer, never in a batch buffer.

The instruction format is:

DWord	Bits	Description
0	31:29	**Instruction Type =** MI_INSTRUCTION = 0h
	28:23	**MI Instruction Opcode =** MI_ARB_CHECK = 05h
	22:0	Reserved: MBZ

1.2.3 MI_BATCH_BUFFER_START

The MI_BATCH_BUFFER_START command format follows:

MI_BATCH_BUFFER_START	
Project:	All
Default Value:	00000000h
Engine:	**Video**

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

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 31h MI_BATCH_BUFFER_START Format: OpCode
	21:13	**Reserved** Project: All Format: MBZ
	12	**Batch Buffer Encrypted Memory Read Enable** Project: All Format: The Command Streamer will request batch buffer data from serpent memory if this bit is enabled. If disabled then the batch buffer will be fetched from non-encrypted memory. Commands in the Table 3-7 Priviledged Commands are not allowed from Encryped Batch Buffers and will be turned into NOOP commands in the command streamer. Any write that is generated from the encrypted batch buffer will write encrypted data.
	11:9	**Reserved** Project: All Format: MBZ

MI_BATCH_BUFFER_START		
	8	**Buffer Security Indicator** Project: All Format: U32 When this command is executed directly from a ring buffer, this field is used to specify the associated batch buffer as a *secure* or *non-secure* buffer. Certain operations (e.g., MI_STORE_DATA_IMM commands) are prohibited within non-secure buffers. See Batch Buffer Protection in the Device Programming Interface chapter of *MI Functions*. When this command is executed from within a batch buffer (i.e., is a "chained" batch buffer command), this field is IGNORED and the next buffer in the chain inherits the initial buffer's security characteristics. If this bit is set, this batch buffer is non-secure and cannot execute privileged commands nor access privileged (GGTT) memory. It will be accessed via the PPGTT. If clear, this batch buffer is secure and will be accessed via the GGTT. Note that MI_STORE_DATA_IMM to non-privileged memory (via the PPGTT) *is* allowed in a non-secure batch buffer. Format = MI_BufferSecurityType 1 = MIBUFFER_NONSECURE 0 = MIBUFFER_SECURE
	7:0	**DWord Length** (Excludes D-Word 0,1) = 0
1	31:2	**Buffer Start Address** Format: Graphics Virtual Address[31:2] FormatDesc **Programming Notes** • A batch buffer initiated with this command must end either with a MI_BATCH_BUFFER_END command or by chaining to another batch buffer with an MI_BATCH_BUFFER_START command. The selection of PPGTT vs. GGTT for the batch buffer is determined by the **Buffer Security Indicator** (bit 8).
	1:0	**Reserved** Project: All Format: MBZ

1.2.4 MI_LOAD_REGISTER_IMM

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 batch buffer then the behavior of this command is controlled by Dword 0, Bit 8 (**Security Indicator**) of the BATCH_BUFFER_START Command. If the batch buffer is non-secure then the command stream converts this command to a NOOP.

The MI_LOAD_REGISTER_IMM command format is:

DWord	Bit	Description
0	31:29	**Command Type =** MI_COMMAND = 0h
	28:23	**MI Command 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.
	7:6	Reserved: MBZ
	5:0	**DWord Length** (Excludes DWord 0,1) = 1.
1	31:23	Reserved: MBZ
	22:2	**Register Offset**: This field specifies bits [22:2] of the offset into the Memory Mapped Register Range (i.e., this field specifies a DWord offset). Format = U30.
	1:0	Reserved: MBZ
2	31:0	**Data DWord.:** This field specifies the DWord value to be written to the targeted location. Format = U32.

1.2.5 MI_NOOP

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).

The MI_NOOP command format is:

DWord	Bit	Description
0	31:29	**Command Type =** MI_COMMAND = 0h
	28:23	**MI Command Opcode =** MI_NOOP = 00h
	22	**Identification Number Register Write Enable:** This field enables the value in the Identification Number field to be written into the MI NOPID register. If disabled, that register is unmodified – making this command an effective “no operation” function. Format = Enable. 1 = Write the NOP_ID register. 0 = Do not write the NOP_ID register.
	21:0	**Identification Number:** This field contains a 22-bit number which can be written to the MI NOPID register. Format = U22.

1.2.6 MI_REPORT_HEAD

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

when the **Per-Process Virtual Address Space Enable** bit is reset:

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

Programming Notes:

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

When the **Per-Process Virtual Address Space Enable** is set, the head pointer will be reported to the PP HW Status Page.

The format of the MI_REPORT_HEAD command is:

DWord	Bit	Description
0	31:29	**Command Type =** MI_COMMAND = 0h
	28:23	**MI Command Opcode =** MI_REPORT_HEAD = 07h
	22:0	Reserved: MBZ

1.2.7 MI_STORE_DATA_IMM

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

Programming Notes:
This command should not be used within a "non-secure" batch buffer to access global virtual space. Doing so will cause the command parser to perform the write with byte enables turned off. This command can be used within ring buffers and/or "secure" batch buffers. If used within a non-secure batch buffer, **Use Global GTT** must be clear.
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.

The MI_STORE_DATA_IMM command format is:

DWord	Bit	Description
0	31:29	**Command Type =** MI_COMMAND = 0h
	28:23	**MI Command Opcode =** MI_STORE_DATA_IMM = 20h
	22	**Use Global GTT.** If set, this command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer. If clear, the PPGTT will be used. 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.
	22:6	Reserved: MBZ
	5:0	**DWord Length** (Excludes DWord 0,1) = 3 for QWord, 2 for DWord
1	31:0	**Reserved: MBZ**
2	31:2	**Address**: This field specifies Bits 31:2 of the Address where the DWord will be stored. As the store address must be DWord-aligned, Bits 1:0 of that address MBZ. This address must be 8B aligned for a store "QW" command. Format = Bits[31:2] of a Graphics Virtual Address
	1:0	Reserved: MBZ
3	31:0	**Data DWord 0:** 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). Format = U32
4	31:0	**Data Word 1:** This field specifies the upper DWord value to be written to the targeted QWord location (DW 1). Format = U32

1.2.8 MI_STORE_DATA_INDEX

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.

The MI_STORE_DATA_INDEX command format is:

DWord	Bit	Description
0	31:29	**Command Type =** MI_COMMAND = 0h
	28:23	**MI Command Opcode =** MI_STORE_DATA_INDEX = 21h
	22	Reserved: MBZ
	21	**Use Per-Process Hardware Status Page**. If this bit is set, this command will index into the per-process hardware status page at offset 20K from the LRCA. If clear, the Global Hardware Status Page will be indexed. This bit will be ignored and treated as <u>set</u> if this command is executed from within a non-secure batch buffer, or if the **Per-Process Virtual Address Space Enable** bit is reset. All other devices: Reserved: MBZ.
	20:8	Reserved: MBZ
	7:0	**DWord Length** (Excludes DWord 0,1) = 2 for QWord
1	31:12	Reserved: MBZ
	11:2	**Offset:** This field specifies the offset (into the hardware status page) to which the data will be written. Note that the first few DWords of this status page are reserved for special-purpose data storage – targeting these reserved locations via this command is UNDEFINED. For a QWord write, the offset is valid down to bit 3 only. Format = U10 zero-based DWord offset into the HW status page. Range = [16, 1023].
	1:0	Reserved: MBZ
2	31:0	**Data DWord 0:** 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). Format = U32
3	31:0	**Data Word 1:** This field specifies the upper DWord value to be written to the targeted QWord location (DW 1). Format = U32

1.2.9 MI_SUSPEND_FLUSH

MI_SUSPEND_FLUSH		
Project: All		**Length Bias:** 1
Blocks MMIO sync flush or any flushes related to VT-d while enabled.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Bh MI_SUSPEND_FLUSH Format: OpCode
	22:1	**Reserved** Project: All Format: MBZ
	0	**Suspend Flush** Project: All Default Value: 0h DefaultVaueDesc Format: Enable FormatDesc This field suspends flush due to sync flush or implicit flush generated during VTD enable, disable and IOTLB invalidation.

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

1.2.10 MI_USER_INTERRUPT

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 =** MI_COMMAND = 0h
	28:23	**MI Command Opcode =** MI_USER_INTERRUPT = 02h
	22:0	Reserved: MBZ

1.2.11 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
	22:20	**Reserved** Project: All Format: MBZ

MI_WAIT_FOR_EVENT		
	19:16	**Condition Code Wait Select** Project: All This field enables a wait for the duration that the corresponding condition code is active. These enable select one of 15 condition codes in the EXCC register, that cause the parser to wait until that condition-code in the EXCC is cleared.

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

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

15:0	**Reserved**	Project: All	Format: MBZ	