Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 6: Command Stream Programming

For the 2014 Intel Atom™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "BayTrail" Platform (ValleyView graphics)

Table of Contents

Graphics 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 processes 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 products include two Command Parsers, each controlling an independent processing engine. These are referred to in this document as the Render Command Parser (RCP) and the Video Codec Command Parser (VCCP).

Valid client values for the Render Command Parser are:

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

Valid client values for the Video Codec Command Parser are:

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

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

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

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

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

Command Header

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	100	XX				

Bits						
TYPE	**31:29**	**28:24**		**23**	**22**	**21:0**
Reserved	101	XX				
Reserved	110	XX				
Reserved	111	XX				

Note: 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).

VCCP Command Header Format

Bits						
TYPE	**31:29**	**28:24**		**23**	**22**	**21:0**
Memory Interface (MI)	000	Opcode 00h – NOP 0Xh – Single DWord Commands 1Xh – Reserved 2Xh – Store Data Commands 3Xh – Ring/Batch Buffer Cmds			Identification No./DWord Count Command Dependent Data 5:0 – DWord Count 5:0 – DWord Count 5:0 – DWord Count	
TYPE	**31:29**	**28:27**	**26:24**	**23:16**		**15:0**
Reserved	011	00	Reserved	Reserved		Reserved
MFX Single DW	011	01	000	Opcode: 0h		0
Reserved	011	01	1XX			
Reserved	011	10	0XX			
AVC State	011	10	100	Opcode: 0h – 4h		DWord Count
AVC Object	011	10	100	Opcode: 8h		DWord Count
VC1 State	011	10	101	Opcode: 0h – 4h		DWord Count
VC1 Object	011	10	101	Opcode: 8h		DWord Count
Reserved	011	10	11X	Reserved		Reserved
Reserved	011	11	Reserved	Reserved		Reserved
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

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

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

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

Memory Interface Commands for RCP

Opcode (28:23)	Command
1 DWord	
00h	MI_NOOP
01h	
02h	MI_USER_INTERRUPT
03h	MI_WAIT_FOR_EVENT
05h	MI_ARB_CHECK
06h	
07h	MI_REPORT_HEAD
08h	MI_ARB_ON_OFF
0Ah	MI_BATCH_BUFFER_END
0Bh	MI_SUSPEND_FLUSH
0Ch	MI_PREDICATE

Opcode (28:23)	Command
0Dh	MI_TOPOLOGY_FILTER
2 Dwords	
10h	Reserved
14h	MI_DISPLAY_FLIP
15h	Reserved
16h	MI_SEMAPHORE_MBOX
17h	Reserved
18h	MI_SET_CONTEXT
1Ah	MI_MATH
1Eh–1Fh	Reserved
Store Data	
20h	MI_STORE_DATA_IMM
21h	MI_STORE_DATA_INDEX
22h	MI_LOAD_REGISTER_IMM
23h	MI_UPDATE_GTT
24h	MI_STORE_REGISTER_MEM
26h	MI_FLUSH_DW
27h	MI_CLFLUSH
28h	MI_REPORT_PERF_COUNT
29h	MI_LOAD_REGISTER_MEM
Ring/Batch Buffer	
30h	Reserved
31h	MI_BATCH_BUFFER_START
32h–35h	Reserved
36h	MI_CONDITIONAL_BATCH_BUFFER_END
37h–3Fh	Reserved

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.

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

Table: 2D Command Map

Opcode (28:22)	Command
00h	Reserved
01h	XY_SETUP_BLT
02h	Reserved
03h	XY_SETUP_CLIP_BLT
04h-10h	Reserved
11h	XY_SETUP_MONO_PATTERN_SL_BLT
12h-23h	Reserved
24h	XY_PIXEL_BLT
25h	XY_SCANLINES_BLT
26h	XY_TEXT_BLT
27h-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

Opcode (28:22)	Command
74h	XY_FULL_IMMEDIATE_PATTERN_BLT
75h	XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT
76h	XY_PAT_CHROMA_BLT
77h	XY_PAT_CHROMA_BLT_IMMEDIATE
78h-7Fh	Reserved

3D Commands

The 3D commands are used to program the graphics pipelines for 3D 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.

For all commands listed in **3D Command Map**, the Pipeline Type (bits 28:27) is 3h, indicating the 3D Pipeline.

Table: 3D Command Map

Opcode Bits 26:24	Sub Opcode Bits 23:16	Command	Definition Chapter
0h	03h	Reserved	
0h	04h	3DSTATE_CLEAR_PARAMS	3D Pipeline
0h	05h	3DSTATE_DEPTH_BUFFER	3D Pipeline
0h	06h	3DSTATE_STENCIL_BUFFER	3D Pipeline
0h	07h	3DSTATE_HIER_DEPTH_BUFFER	3D Pipeline
0h	08h	3DSTATE_VERTEX_BUFFERS	Vertex Fetch
0h	09h	3DSTATE_VERTEX_ELEMENTS	Vertex Fetch
0h	0Ah	3DSTATE_INDEX_BUFFER	Vertex Fetch
0h	0Bh	3DSTATE_VF_STATISTICS	Vertex Fetch
0h	0Ch	Reserved	
0h	0Dh	3DSTATE_VIEWPORT_STATE_POINTERS	3D Pipeline
0h	0Eh	3DSTATE_CC_STATE_POINTERS	3D Pipeline
0h	10h	3DSTATE_VS	Vertex Shader
0h	11h	3DSTATE_GS	Geometry Shader

Opcode Bits 26:24	Sub Opcode Bits 23:16	Command	Definition Chapter
0h	12h	3DSTATE_CLIP	Clipper
0h	13h	3DSTATE_SF	Strips & Fans
0h	14h	3DSTATE_WM	Windower
0h	15h	3DSTATE_CONSTANT_VS	Vertex Shader
0h	16h	3DSTATE_CONSTANT_GS	Geometry Shader
0h	17h	3DSTATE_CONSTANT_PS	Windower
0h	18h	3DSTATE_SAMPLE_MASK	Windower
0h	19h	3DSTATE_CONSTANT_HS	Hull Shader
0h	1Ah	3DSTATE_CONSTANT_DS	Domain Shader
0h	1Bh	3DSTATE_HS	Hull Shader
0h	1Ch	3DSTATE_TE	Tesselator
0h	1Dh	3DSTATE_DS	Domain Shader
0h	1Eh	3DSTATE_STREAMOUT	HW Streamout
0h	1Fh	3DSTATE_SBE	Setup
0h	20h	3DSTATE_PS	Pixel Shader
0h	21h	3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP	Strips & Fans
0h	22h	Reserved	
0h	23h	3DSTATE_VIEWPORT_STATE_POINTERS_CC	Windower
0h	24h	3DSTATE_BLEND_STATE_POINTERS	Pixel Shader
0h	25h	3DSTATE_DEPTH_STENCIL_STATE_POINTERS	Pixel Shader
0h	26h	3DSTATE_BINDING_TABLE_POINTERS_VS	Vertex Shader

Opcode Bits 26:24	Sub Opcode Bits 23:16	Command	Definition Chapter
0h	27h	3DSTATE_BINDING_TABLE_POINTERS_HS	Hull Shader
0h	28h	3DSTATE_BINDING_TABLE_POINTERS_DS	Domain Shader
0h	29h	3DSTATE_BINDING_TABLE_POINTERS_GS	Geometry Shader
0h	2Ah	3DSTATE_BINDING_TABLE_POINTERS_PS	Pixel Shader
0h	2Bh	3DSTATE_SAMPLER_STATE_POINTERS_VS	Vertex Shader
0h	2Ch	3DSTATE_SAMPLER_STATE_POINTERS_HS	Hull Shader
0h	2Dh	3DSTATE_SAMPLER_STATE_POINTERS_DS	Domain Shader
0h	2Eh	3DSTATE_SAMPLER_STATE_POINTERS_GS	Geometry Shader
0h	2Fh	Reserved	
0h	30h	3DSTATE_URB_VS	Vertex Shader
0h	31h	3DSTATE_URB_HS	Hull Shader
0h	32h	3DSTATE_URB_DS	Domain Shader
0h	33h	3DSTATE_URB_GS	Geometry Shader
0h	48h-4Bh	Reserved	
0h	4Ch	3DSTATE_WM_CHROMA_KEY	Windower
0h	4Dh	3DSTATE_PS_BLEND	Windower
0h	4Eh	3DSTATE_WM_DEPTH_STENCIL	Windower
0h	4Fh	3DSTATE_PS_EXTRA	Windower
0h	50h	3DSTATE_RASTER	Strips & Fans
0h	51h	3DSTATE_SBE_SWIZ	Strips & Fans
0h	52h	3DSTATE_WM_HZ_OP	Windower
0h	53h	3DSTATE_INT (internally generated state)	3D Pipeline
0h	56h-FFh	Reserved	
1h	00h	3DSTATE_DRAWING_RECTANGLE	Strips & Fans
1h	02h	3DSTATE_SAMPLER_PALETTE_LOAD0	Sampling Engine
1h	03h	Reserved	
1h	04h	3DSTATE_CHROMA_KEY	Sampling Engine

Opcode Bits 26:24	Sub Opcode Bits 23:16	Command	Definition Chapter
1h	05h	Reserved	
1h	06h	3DSTATE_POLY_STIPPLE_OFFSET	Windower
1h	07h	3DSTATE_POLY_STIPPLE_PATTERN	Windower
1h	08h	3DSTATE_LINE_STIPPLE	Windower
1h	0Ah	3DSTATE_AA_LINE_PARAMS	Windower
1h	0Bh	3DSTATE_GS_SVB_INDEX	Geometry Shader
1h	0Ch	3DSTATE_SAMPLER_PALETTE_LOAD1	Sampling Engine
1h	0Dh	3DSTATE_MULTISAMPLE	Windower
1h	0Eh	3DSTATE_STENCIL_BUFFER	Windower
1h	0Fh	3DSTATE_HIER_DEPTH_BUFFER	Windower
1h	10h	3DSTATE_CLEAR_PARAMS	Windower
1h	11h	3DSTATE_MONOFILTER_SIZE	Sampling Engine
1h	12h	3DSTATE_PUSH_CONSTANT_ALLOC_VS	Vertex Shader
1h	13h	3DSTATE_PUSH_CONSTANT_ALLOC_HS	Hull Shader
1h	14h	3DSTATE_PUSH_CONSTANT_ALLOC_DS	Domain Shader
1h	15h	3DSTATE_PUSH_CONSTANT_ALLOC_GS	Geometry Shader
1h	16h	3DSTATE_PUSH_CONSTANT_ALLOC_PS	Pixel Shader
1h	17h	3DSTATE_SO_DECL_LIST	HW Streamout
1h	18h	3DSTATE_SO_BUFFER	HW Streamout
1h	1Ch	3DSTATE_SAMPLE_PATTERN	Windower
1h	1Dh	3DSTATE_URB_CLEAR	3D Pipeline
1h	1Eh-FFh	Reserved	
2h	00h	PIPE_CONTROL	3D Pipeline
2h	01h-FFh	Reserved	
3h	00h	3DPRIMITIVE	Vertex Fetch
3h	01h-FFh	Reserved	
4h-7h	00h-FFh	Reserved	

Pipeline Type (28:27)	Opcode	Sub Opcode	Command	Definition Chapter
Common (pipelined)	**Bits 26:24**	**Bits 23:16**		
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	SWTESS BASE ADDRESS	3D Pipeline
0h	1h	04h	GPGPU CSR BASE ADDRESS	Graphics Processing Engine
0h	1h	04h–FFh	**Reserved**	n/a
Reserved	**Bits 26:24**	**Bits 23:16**		
0h	2h–7h	XX	**Reserved**	n/a

MFX Commands

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

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

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

Pipeline Type (28:27)	Opcode (26:24)	Subop A (23:21)	Subop B (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
2h	1h	2h	2h	MFC_AVC_FQM_STATE	MFX	IMAGE	N/A
2h	1h	2h	3-7h	Reserved	N/A	N/A	N/A
2h	1h	2h	8h	MFC_AVC_PAK_INSERT_OBJECT	MFX	N/A	N/A
2h	1h	2h	9h	MFC_AVC_PAK_OBJECT	MFX	N/A	Yes
2h	1h	2h	A-1Fh	Reserved	N/A	N/A	N/A
2h	1h	2h	0-1Fh	Reserved	N/A	N/A	N/A
VC1 Common							
2h	2h	0h	0h	MFX_VC1_PIC_STATE	MFX	IMAGE	N/A
2h	2h	0h	1h	MFX_VC1_PRED_PIPE_STATE	MFX	IMAGE	N/A
2h	2h	0h	2h	MFX_VC1_DIRECTMODE_STATE	MFX	SLICE	N/A
2h	2h	0h	2-1Fh	Reserved	N/A	N/A	N/A
VC1 Dec							
2h	2h	1h	0-7h	Reserved	N/A	N/A	N/A
2h	2h	1h	8h	MFD_VC1_BSD_OBJECT	MFX	N/A	Yes
2h	2h	1h	9-1Fh	Reserved	N/A	N/A	N/A
VC1 Enc							
2h	2h	2h	0-1Fh	Reserved	N/A	N/A	N/A
MPEG2Common							
2h	3h	0h	0h	MFX_MPEG2_PIC_STATE	MFX	IMAGE	N/A
2h	3h	0h	1h	MFX_MPEG2_QM_STATE	MFX	IMAGE	N/A
2h	3h	0h	2-1Fh	Reserved	N/A	N/A	N/A
MPEG2 Dec							
2h	3h	1h	1-7h	Reserved	N/A	N/A	N/A
2h	3h	1h	8h	MFD_MPEG2_BSD_OBJECT	MFX	N/A	Yes
2h	3h	1h	9-1Fh	Reserved	N/A	N/A	N/A
MPEG2 Enc							
2h	3h	2h	0-1Fh	Reserved	N/A	N/A	N/A
The Rest							
2h	4-5h, 7h	x	x	Reserved	N/A	N/A	N/A

Blitter Engine Command Interface

BCS_RINGBUF—Ring Buffer Registers

Following is a list of ring buffer registers:

RING_BUFFER_TAIL - Ring Buffer Tail

RING_BUFFER_HEAD - Ring Buffer Head

RING_BUFFER_START - Ring Buffer Start

RING_BUFFER_CTL - Ring Buffer Control

UHPTR - Pending Head Pointer Register

Blitter Engine Command Interface

BCS_RINGBUF—Ring Buffer Registers

Following is a list of ring buffer registers:

RING_BUFFER_TAIL - Ring Buffer Tail

RING_BUFFER_HEAD - Ring Buffer Head

RING_BUFFER_START - Ring Buffer Start

RING_BUFFER_CTL - Ring Buffer Control

UHPTR - Pending Head Pointer Register

BLT Watchdog Timer Registers

These are the Watchdog Timer registers:

BCS_CTR_THRSH - BCS Watchdog Counter Threshold

PR_CTR_THRSH - Watchdog Counter Threshold

PR_CTR_CTL - Watchdog Counter Control

BLT Interrupt Control Registers

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

Bit Definition for Interrupt Control Registers

Bit	Description
31:30	**Reserved. MBZ:** These bits may be assigned to interrupts on future products/steppings.
29	**Reserved**
28:27	**Reserved. MBZ**
26	**MI_FLUSH_DW Notify Interrupt:** The Pipe Control packet (Fences) specified in *3D pipeline* document may optionally generate an Interrupt. The Store QW associated with a fence is completed ahead of the interrupt.
25	Blitter Command Parser Master Error: When this status bit is set, it indicates that the hardware has detected an error. It is set by the device upon an error condition and cleared by a CPU write of a one to the appropriate bit contained in the Error ID register followed by a write of a one to this bit in the IIR. Further information on the source of the error comes from the "Error Status Register" which along with the "Error Mask Register" determine which error conditions will cause the error status bit to be set and the interrupt to occur. Page Table Error: Indicates a page table error. Instruction Parser Error: The Blitter Instruction Parser encounters an error while parsing an instruction.
24	Sync Status: This bit is set when the Instruction Parser completes a flush with the sync enable bit active in the INSTPM register. The event will happen after all the blitter engines are flushed. The HW Status DWord write resulting from this event will cause the CPU's view of graphics memory to be coherent as well (flush and invalidate the blitter cache). It is the driver's responsibility to clear this bit before the next sync flush with HWSP write enabled.
23	**Reserved. MBZ**
22	Blitter Command Parser User Interrupt: This status bit is set when an MI_USER_INTERRUPT instruction is executed on the Render Command Parser. Note that instruction execution is not halted and proceeds normally. A mechanism such as an MI_STORE_DATA instruction is required to associate a particular meaning to a user interrupt.
21:0	**Reserved. MBZ**

BCS_HWSTAM - BCS Hardware Status Mask Register

BCS_IMR - BCS Interrupt Mask Register

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

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

The following table describes the Hardware-Detected Error bits:

Table: Hardware-Detected Error Bits

BCS Hardware-Detected Error Bit Definitions

Following are the the EIR, EMR and ESR registers:

BCS_EIR - BCS Error Identity Register

BCS_EMR - BCS Error Mask Register

BCS_ESR - BCS Error Status Register

BLT Logical Context Support

Following are the Logical Context Support Registers:

BB_ADDR - Batch Buffer Head Pointer Register

BCS_SYNC_FLIP_STATUS - BCS Wait for event and Display flip flags Register

SYNC_FLIP_STATUS - Wait For Event and Display Flip Flags Register

SYNC_FLIP_STATUS_1 - Wait For Event and Display Flip Flags Register 1

RING_BUFFER_HEAD_PREEMPT_REG - RING_BUFFER_HEAD_PREEMPT_REG

BLT Mode Registers

Following are BLT Mode Registers:

BCS_CXT_SIZE - BCS Context Sizes

BCS_MI_MODE - BCS Mode Register for Software Interface

BLT_MODE - Blitter Mode Register

BCS_INSTPM - BCS Instruction Parser Mode Register

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

Programming Notes:

- All Reserved bits are implemented.

BCS_EXCC - BCS Execute Condition Code Register

BRSYNC - Blitter/Render Semaphore Sync Register

BVSYNC - Blitter/Video Semaphore Sync Register

Programming Note: If this register is written, a workload must subsequently be dispatched to the render command streamer.

HWS_PGA - Hardware Status Page Address Register

Hardware Status Page Layout

MI Commands for Blitter Engine

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.

MI_NOOP

MI_ARB_CHECK

MI_ARB_ON_OFF

MI_BATCH_BUFFER_START

MI_BATCH_BUFFER_END

MI_DISPLAY_FLIP

MI_FLUSH_DW

MI_REPORT_HEAD

MI_STORE_DATA_IMM

MI_STORE_DATA_INDEX

MI_LOAD_REGISTER_IMM

MI_LOAD_REGISTER_MEM

MI_STORE_REGISTER_MEM

MI_USER_INTERRUPT

MI_WAIT_FOR_EVENT

MI_SEMAPHORE_MBOX

Render Engine Command Interface

Render Engine Command Streamer (RCS)

The RCS (Render Command Streamer) unit primarily serves as the software programming interface between the O/S driver and the Render Engine. It is responsible for fetching, decoding, and dispatching of data packets (3D/Media Commands with the header DWord removed) to the front end interface module of Render Engine.

Its logic functions include:

- MMIO register programming interface.
- DMA action for fetching of ring data from memory.
- Management of the Head pointer for the Ring Buffer.
- Decode of ring data and sending it to the appropriate destination: 3D (Vertex Fetch Unit) & GPGPU.
- Handling of user interrupts.
- Flushing the 3D and GPGPU Engine.
- Handle NOP.

The register programming bus is a DWord interface bus that is driven by the configuration master. The RCS unit only claims memory mapped I/O cycles that are targeted to its range of 0x2000 to 0x27FF. The Gx and MFX Engines use semaphore to synchronize their operations.

RCS operates completely independent of the MFx CS.

The simple sequence of events is as follows: a ring (say PRB0) is programmed by a memory-mapped register write cycle. The DMA inside RCS is kicked off. The DMA fetches commands from memory based on the starting address and head pointer. The DMA requests cache lines from memory (one cacheline CL at a time). There is guaranteed space in the DMA FIFO (8 CL deep) for data coming back from memory. The DMA control logic has copies of the head pointer and the tail pointer. The DMA increments the head pointer after making requests for ring commands. Once the DMA copy of the head pointer becomes equal to the tail pointer, the DMA stops requesting.

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

RINGBUF — Ring Buffer Registers

See the "Device Programming Environment" chapter for detailed information on these registers.

RING_BUFFER_TAIL - Ring Buffer Tail

RING_BUFFER_HEAD - Ring Buffer Head

RING_BUFFER_START - Ring Buffer Start

RING_BUFFER_CTL - Ring Buffer Control

UHPTR - Pending Head Pointer Register

Render Watchdog Timer Registers

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

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

PR_CTR_CTL - Watchdog Counter Control

PR_CTR_THRSH - Watchdog Counter Threshold

PR_CTR - Render Watchdog Counter

Render Interrupt Control Registers

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

Bit Definition for Interrupt Control Registers

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

Bit	Interrupt Bit	ISR bit Reporting via Hardware Status Write (when unmasked via HWSTAM)
9	**Performance Monitoring Buffer Half-Full Interrupt**	Set when event occurs, cleared when event cleared
8	**Reserved**	
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	**L3 Parity interrupt**	
4	PIPE_CONTROL packet - Notify Enable	0
3	Master Error	Set when error occurs, cleared when error cleared
2	Sync Status	Toggled every SyncFlush Event
1		
0	User Interrupt	0

HWSTAM - Hardware Status Mask Register

IMR - Interrupt Mask Register

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

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

The following table describes the Hardware-Detected Error bits:

Table: Hardware-Detected Error Bits

Hardware-Detected Error Bit Definitions

Following are the the EIR, EMR and ESR registers:

EIR - Error Identity Register

EMR - Error Mask Register

ESR - Error Status Register

Render Logical Context Support

Following are the Logical Context Support Registers:

BB_ADDR - Batch Buffer Head Pointer Register

RCS_BB_STATE - RCS Batch Buffer State Register

CCID - Current Context Register

SYNC_FLIP_STATUS - Wait For Event and Display Flip Flags Register

SYNC_FLIP_STATUS_1 - Wait For Event and Display Flip Flags Register 1

Context Save Registers

Following are the Context Save Registers:

RING_BUFFER_HEAD_PREEMPT_REG - RING_BUFFER_HEAD_PREEMPT_REG

Mode Registers

Following are the Mode Registers:

INSTPM - Instruction Parser Mode Register

EXCC - Execute Condition Code Register

NOPID - NOP Identification Register

RVSYNC - Render/Video Semaphore Sync Register

RBSYNC - Render/Blitter Semaphore Sync Register

HWS_PGA - Hardware Status Page Address Register

Hardware Status Page Layout

MI Commands for Render Engine

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.

Video Command Streamer (VCS)

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

Its logic functions include:

- MMIO register programming interface.
- Management of the Head pointer for the Ring Buffer.
- Decode of ring data and sending it to the appropriate destination: AVC, VC1, or MPEG2 engine.
- Handling of user interrupts.
- Flushing the MFX Engine.
- Handle NOP.

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

VCS operates completely independent of the Gx CS.

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

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

Video Command Streamer (VCS)

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

Its logic functions include:

- MMIO register programming interface.
- Management of the Head pointer for the Ring Buffer.
- Decode of ring data and sending it to the appropriate destination: AVC, VC1, or MPEG2 engine.
- Handling of user interrupts.
- Flushing the MFX Engine.
- Handle NOP.

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

VCS operates completely independent of the Gx CS.

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

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

VCS_RINGBUF—Ring Buffer Registers

RING_BUFFER_TAIL - Ring Buffer Tail

RING_BUFFER_HEAD - Ring Buffer Head

RING_BUFFER_START - Ring Buffer Start

RING_BUFFER_CTL - Ring Buffer Control

UHPTR - Pending Head Pointer Register

Watchdog Timer Registers

The following registers are defined as Watchdog Timer registers:

VCS_CNTR - VCS Counter for the bit stream decode engine

VCS_THRSH - VCS Threshold for the counter of bit stream decode engine

Interrupt Control Registers

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

Bit Definition for Interrupt Control Registers

Bit	Description
31:21	**Reserved. MBZ:** These bits may be assigned to interrupts on future products/steppings.
20	**Reserved**
19	**Page Fault:** This bit is set whenever there is a pending page or directory fault. This bit is set whenever there is a pending page or directory fault in Video command streamer.
18	**Timeout Counter Expired:** Set when the VCS timeout counter has reached the timeout thresh-hold value.
17	**Reserved**
16	**MI_FLUSH_DW Notify Interrupt:** The Pipe Control packet (Fences) specified in *3D pipeline* document may optionally generate an Interrupt. The Store QW associated with a fence is completed ahead of the interrupt.
15	**Video Command Parser Master Error:** When this status bit is set, it indicates that the hardware has detected an error. It is set by the device upon an error condition and cleared by a CPU write of a one to the appropriate bit contained in the Error ID register followed by a write of a one to this bit in the IIR. Further information on the source of the error comes from the "Error Status Register" which along with the "Error Mask Register" determine which error conditions will cause the error status bit to be set and the interrupt to occur. **Page Table Error:** Indicates a page table error. **Instruction Parser Error**: The Video Instruction Parser encounters an error while parsing an instruction.
14	**Sync Status:** This bit is set when the Instruction Parser completes a flush with the sync enable bit active in the INSTPM register. The event will happen after all the MFX engines are flushed. The HW Status DWord write resulting from this event will cause the CPU's view of graphics memory to be coherent as well (flush and invalidate the MFX cache).It is the driver's responsibility to clear this bit before the next sync flush with HWSP write enabled
13	**Reserved: MBZ**
12	**Video Command Parser User Interrupt:** This status bit is set when an MI_USER_INTERRUPT instruction is executed on the Video Command Parser. Note that instruction execution is not halted and proceeds normally. A mechanism such as an MI_STORE_DATA instruction is required to associate a particular meaning to a user interrupt.
11:0	**Reserved:** MBZ

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

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

VCS_HWSTAM - VCS Hardware Status Mask Register

VCS_IMR - VCS Interrupt Mask Register

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

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

The following table describes the Hardware-Detected Error bits:

Hardware-Detected Error Bits

VCS Hardware-Detected Error Bit Definitions

VCS_EIR - VCS Error Identity Register

VCS_EMR - VCS Error Mask Register

VCS_ESR - VCS Error Status Register

Logical Context Support

This section contains the registers for Logical Context Support.

BB_STATE - Batch Buffer State Register

BB_ADDR - Batch Buffer Head Pointer Register

Mode Registers

Following are Mode Registers:

BBA_LEVEL2 - 2nd Level Batch Buffer Address

VCS_CXT_SIZE - VCS Context Sizes

VCS_MI_MODE - VCS Mode Register for Software Interface

MFX_MODE - Video Mode Register

VCS_INSTPM - VCS Instruction Parser Mode Register

VBSYNC - Video/Blitter Semaphore Sync Register

VRSYNC - Video/Render Semaphore Sync Register

HWS_PGA - Hardware Status Page Address Register

Hardware Status Page Layout

Registers in Media Engine

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

GAC PWR CTX STORAGE REGISTERS

Following are GAC PWR CTX STORAGE Registers:

GFX_PEND_TLB - TLBPEND Control Register

GAC_ARB_CTL_REG - GAC_GAB Arbitration Counters Register 1

GAC_ERROR - Media Arbiter Error Report Register

GFX TLB In Use Virtual Address Registers

TLB064_VA - TLB064_VA Virtual Page Address Registers

TLB132_VA - TLB132_VA Virtual Page Address Registers

TLB232_VA - TLB232_VA Virtual Page Address Registers

TLB304_VA - TLB304_VA Virtual Page Address Registers

MTTLB064_VLD0 - Valid Bit Vector 0 for TLB064

MTTLB064_VLD1 - Valid Bit Vector 1 for TLB064

MTTLB132_VLD0 - Valid Bit Vector 0 for TLB132

MTTLB132_VLD1 - Valid Bit Vector 1 for TLB132

MTTLB232_VLD0 - Valid Bit Vector 0 for TLB232

MTTLB232_VLD1 - Valid Bit Vector 1 for TLB232

MTTLB304_VLD0 - Valid Bit Vector 0 for TLB304

MTTLB304_VLD1 - Valid Bit Vector 1 for TLB304

GFX Pending TLB Cycles Information Registers

The following registers contain information about cycles that did not complete their TLB translation.

Information is organized as 64 entries, where each entry has a valid and ready bit, collapsed into separate registers.

VCS_TLBPEND_VLD0 - VCS Valid Bit Vector 0 for TLBPEND Registers

VCS_TLBPEND_VLD1 - VCS Valid Bit Vector 1 for TLBPEND Registers

VCS_TLBPEND_RDY0 - VCS Ready Bit Vector 0 for TLBPEND Registers

VCS_TLBPEND_RDY1 - VCS Ready Bit Vector 1 for TLBPEND Registers

VCS_TLBPEND_SEC0 - VCS Section 0 of TLBPEND Entry

VCS_TLBPEND_SEC1 - VCS Section 1 of TLBPEND Entry

VCS_TLBPEND_SEC2 - VCS Section 2 of TLBPEND Entry

VCS_TIMESTAMP - VCS Reported Timestamp Count

Memory Interface Commands for Video Codec Engine

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

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

MI_ARB_CHECK

MI_ARB_ON_OFF

MI_BATCH_BUFFER_END

MI_CONDITIONAL_BATCH_BUFFER_END

MI_BATCH_BUFFER_START

MI_FLUSH_DW

MI_LOAD_REGISTER_IMM

MI_NOOP

MI_REPORT_HEAD

MI_SEMAPHORE_MBOX

MI_STORE_REGISTER_MEM

MI_STORE_DATA_IMM

MI_STORE_DATA_INDEX

MI_SUSPEND_FLUSH

MI_USER_INTERRUPT

MI_WAIT_FOR_EVENT

MI_LOAD_REGISTER_MEM

Preemption

Preemption is a means by which HW is instructed to stop executing an ongoing workload and switch to the new workload submitted. Preemption flows are different based on the mode of scheduling.

Ring Buffer Scheduling

In Ring Buffer mode of scheduling SW triggers preemption by programming UHPTR (Updated Head Pointer Register) register with a valid head pointer. UHPTR contains head pointer and head pointer valid bit, head pointer is valid only when the head pointer valid bit is set.

HW triggers preemption on a preemptable command on detecting Head Pointer Valid bit asserted in the UHPTR register. Following preemption HW updates it current head pointer with the Head Pointer from the UHPTR and starts execution i.e all the commands from current head pointer to the updated head pointer are skipped by HW. HW samples the head pointer and the batch buffer address on preemption and updates them to the RING_BUFEFR_HEAD_PREEMPT_REG and BB_PREEMPT_ADDR respectively. RING_BUFFER_HEAD_PREEMPT_REG and BB_PREEMPT_ADDR provide the graphics memory address of the preemptable command on which last preemption has occurred. HW resets the head pointer valid bit in UHPTR upon completion of preemption.

Programming Notes:

Preemption is not supported for Media Workloads. Hence preemption can be achieved only on Command Buffer boundaries. Media Command Buffers must be bracketed with MI_ARB_OFF and MI_ARB_ON command to avoid preemption of media command buffers.

Example:

Ring Buffer

.

.

.

MI_ARB_ON_OFF → OFF

MI_BATCH_START – Media Workload

MI_ARB_ON_OFF → ON

MI_ARB_CHK → preemptable command outside media command buffer.

.

.

End Ring Buffer

The following table lists the Preemptable Commands in Ring Buffer mode of scheduling:

Preemptable Commands → Engine	MI_ARB_CHECK
Render	AP*
Blitter	AP*
Media	AP*
VideoEnhancement	AP*

AP*: Allow Preemption on UHPTR valid.