Intel® Open Source HD Graphics, Intel Iris™ Graphics, and Intel Iris™ Pro Graphics

Programmer's Reference Manual

For the 2015 - 2016 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Skylake" Platform

Volume 6: Command Stream Programming

May 2016, Revision 1.0

Table of Contents

Graphics Command Formats 1
Command Header 2
Memory Interface Commands 5
2D Commands 7
3D Commands 8
VEBOX Commands 13
MFX Commands 13
Scheduling 15
RINGBUF — Ring Buffer Registers 15
Command Stream Virtual Memory Control 15
Execlists 15
Execlist Structure 16
Overall Context Layout 17
Context Layout 17
Ring Context 17
Ring Buffer 17
Context Descriptor Format 18
Logical Ring Context Format 18
Context Status 20
Render Engine Command Streamer (RCS) 23
Batch Buffer Privilege Register 24
Mode Registers 24
Logical Context Support 24
Context Save Registers 25
MI Commands for Render Engine 25
Watchdog Timer Registers 26
Interrupt Control Registers 27
Bit Definition for Interrupt Control Registers: 27
Hardware-Detected Error Bit Definitions (for EIR EMR ESR) 28
Blitter Engine Command Streamer (BCS) 28
Logical Context Support 29
Mode Registers 30

MI Commands for Blitter Engine....30

Video Command Streamer (VCS) 31

Watchdog Timer Registers 32

Logical Context Support....33

Mode Registers 34

Registers in Media Engine 34

GFX Pending TLB Cycles Information Registers 34

Memory Interface Commands for Video Codec Engine....35

Video Enhancement Engine Command Interface 36

VECS_RINGBUF — Ring Buffer Registers....36

Logical Context Support....36

Mode Registers 37

MI Commands for Video Enhancement Engine....37

Watchdog Timers....38

The Per-Process Hardware Status Page 39

Preemption 39

Ring Buffer Scheduling 40

ExecList Scheduling 41

Command Streamer (CS) ALU Programming 43

MI Commands for Graphics Processing Engines....43

User Mode Privileged Commands....44

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.

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

Render 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, 010	Opcode – 11111	23:19 Sub Opcode 00h – 01h	18:16 Reserved	15:0 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)[1]	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)[1]	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				
Reserved	101	XX				
Reserved	110	XX				

Notes:

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

Video Command Header Format

Bits					
TYPE	**31:29**	**28:24**	**23**	**22**	**21:0**
Memory Interface (MI)	000	Opcode 00h – NOP 0Xh – Single DWord Commands 1Xh – Reserved 2Xh – Store Data Commands 3Xh – Ring/Batch Buffer Cmds		Identification No./DWord Count Command Dependent Data 5:0 – DWord Count 5:0 – DWord Count 5:0 – DWord Count	

TYPE	31:29	28:27	26:24	23:16	15:0
Reserved	011	00	XXX	XX	
MFX Single DW	011	01	000	Opcode: 0h	0
Reserved	011	01	1XX		
Reserved	011	10	0XX		
AVC State	011	10	100	Opcode: 0h – 4h	DWord Count
AVC Object	011	10	100	Opcode: 8h	DWord Count
VC1 State	011	10	101	Opcode: 0h – 4h	DWord Count
VC1 Object	011	10	101	Opcode: 8h	DWord Count
Reserved	011	10	11X		
Reserved	011	11	XXX		

TYPE	31:29	28:27	26:24	23:21	20:16	15:0
MFX Common	011	10	000	000	subopcode	DWord Count
Reserved	011	10	000	001-111	subopcode	DWord Count
AVC Common	011	10	001	000	subopcode	DWord Count
AVC Dec	011	10	001	001	subopcode	DWord Count
AVC Enc	011	10	001	010	subopcode	DWord Count
Reserved	011	10	001	011-111	subopcode	DWord Count
Reserved (for VC1 Common)	011	10	010	000	subopcode	DWord Count
VC1 Dec	011	10	010	001	subopcode	DWord Count
Reserved (for VC1 Enc)	011	10	010	010	subopcode	DWord Count
Reserved	011	10	010	011-111	subopcode	DWord Count
Reserved (MPEG2 Common)	011	10	011	000	subopcode	DWord Count
MPEG2 Dec	011	10	011	001	subopcode	DWord Count
Reserved (for MPEG2 Enc)	011	10	011	010	subopcode	DWord Count
Reserved	011	10	011	011-111	subopcode	DWord Count
Reserved	011	10	100-111	XXX		

Video Enhancement 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, 010				

TYPE	31:29	28:27	26:24	23:21	20:16	15:12	11:0
VEBOX (Parallel Video Pipe)	011	10: Pipeline 00: Reserved 01: Reserved 11: Reserved	Command Opcode – 100	Sub Opcode A	Sub Opcode B	Reserved	Dword Count

Blitter 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, 011				

TYPE	31:29	28:22	21:9	8:0
Blitter (2D)	010	Command Opcode	Command Dependent Data	Dword Count

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

Memory Interface Commands for RCP

Opcode (28:23)	Command	Pipes
1 DWord		
00h	MI_NOOP	All
01h	MI_SET_PREDICATE	Render
02h	MI_USER_INTERRUPT	All
03h	MI_WAIT_FOR_EVENT	All
05h	MI_ARB_CHECK	All
06h	MI_RS_CONTROL	Render
07h	MI_REPORT_HEAD	All
08h	MI_ARB_ON_OFF	All except Blitter
09h	MI_URB_ATOMIC_ALLOC	Render
0Ah	MI_BATCH_BUFFER_END	All
0Bh	MI_SUSPEND_FLUSH	All
0Ch	MI_PREDICATE	Render
0Dh	MI_TOPOLOGY_FILTER	Render
0Fh	MI_RS_CONTEXT	Render
2+ DWord		
10h	Reserved	
14h	MI_DISPLAY_FLIP	Render and Blitter
15h	Reserved	
16h	MI_SEMAPHORE_MBOX	All
17h	Reserved	
18h	MI_SET_CONTEXT	Render
1Ah	MI_MATH	All
1Bh	MI_SEMAPHORE_SIGNAL	All

Opcode (28:23)	Command	Pipes
1 DWord		
1Ch	MI_SEMAPHORE_WAIT	All
1Dh	MI_FORCE_WAKEUP	All except Render
1Fh	Reserved	
Store Data		
20h	MI_STORE_DATA_IMM	All
21h	MI_STORE_DATA_INDEX	All
22h	MI_LOAD_REGISTER_IMM	All
23h	MI_UPDATE_GTT	All
24h	MI_STORE_REGISTER_MEM	All
26h	MI_FLUSH_DW	All except Render
27h	MI_CLFLUSH	Render
29h	MI_LOAD_REGISTER_MEM	All
2Ah	MI_LOAD_REGISTER_REG	All
2Bh	MI_RS_STORE_DATA_IMM	Render
2Ch	MI_LOAD_URB_MEM	Render
2Dh	MI_STORE_URB_MEM	Render
2Eh	MI_MEM_TO_MEM	All
2Fh	MI_ATOMIC	All
Ring/Batch Buffer		
30h	Reserved	
31h	MI_BATCH_BUFFER_START	Render
32h-35h	Reserved	
36h	MI_CONDITIONAL_BATCH_BUFFER_END	All
37h-38h	Reserved	
39h-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*.

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

Opcode (28:22)	Command
72h	XY_PAT_BLT_IMMEDIATE
73h	XY_SRC_COPY_CHROMA_BLT
74h	XY_FULL_IMMEDIATE_PATTERN_BLT
75h	XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT
76h	XY_PAT_CHROMA_BLT
77h	XY_PAT_CHROMA_BLT_IMMEDIATE
78h-7Fh	Reserved

3D 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.

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	3DSTATE_VF	Vertex Fetch
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
0h	12h	3DSTATE_CLIP	Clipper
0h	13h	3DSTATE_SF	Strips & Fans
0h	14h	3DSTATE_WM	Windower
0h	15h	3DSTATE_CONSTANT_VS	Vertex Shader

Opcode Bits 26:24	Sub Opcode Bits 23:16	Command	Definition Chapter
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	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
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	34h	3DSTATE_GATHER_CONSTANT_VS	Vertex Shader
0h	35h	3DSTATE_GATHER_CONSTANT_GS	Geometry Shader
0h	36h	3DSTATE_GATHER_CONSTANT_HS	Hull Shader
0h	37h	3DSTATE_GATHER_CONSTANT_DS	Domain Shader
0h	38h	3DSTATE_GATHER_CONSTANT_PS	Pixel Shader
0h	39h	3DSTATE_DX9_CONSTANTF_VS	Vertex Shader
0h	3Ah	3DSTATE_DX9_CONSTANTF_PS	Pixel Shader

Opcode Bits 26:24	Sub Opcode Bits 23:16	Command	Definition Chapter
0h	3Bh	3DSTATE_DX9_CONSTANTI_VS	Vertex Shader
0h	3Ch	3DSTATE_DX9_CONSTANTI_PS	Pixel Shader
0h	3Dh	3DSTATE_DX9_CONSTANTB_VS	Vertex Shader
0h	3Eh	3DSTATE_DX9_CONSTANTB_PS	Pixel Shader
0h	3Fh	3DSTATE_DX9_LOCAL_VALID_VS	Vertex Shader
0h	40h	3DSTATE_DX9_LOCAL_VALID_PS	Pixel Shader
0h	41h	3DSTATE_DX9_GENERATE_ACTIVE_VS	Vertex Shader
0h	42h	3DSTATE_DX9_GENERATE_ACTIVE_PS	Pixel Shader
0h	43h	3DSTATE_BINDING_TABLE_EDIT_VS	Vertex Shader
0h	44h	3DSTATE_BINDING_TABLE_EDIT_GS	Geometry Shader
0h	45h	3DSTATE_BINDING_TABLE_EDIT_HS	Hull Shader
0h	46h	3DSTATE_BINDING_TABLE_EDIT_DS	Domain Shader
0h	47h	3DSTATE_BINDING_TABLE_EDIT_PS	Pixel Shader
0h	48h	3DSTATE_VF_HASHING	Vertex Fetch
0h	49h	3DSTATE_VF_INSTANCING	Vertex Fetch
0h	4Ah	3DSTATE_VF_SGVS	Vertex Fetch
0h	4Bh	3DSTATE_VF_TOPOLOGY	Vertex Fetch
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	54h	3DSTATE_RS_CONSTANT_POINTER	Resource Streamer
0h	55h	3DSTATE_VF_COMPONENT_PACKING	Vertex Fetch
0h	56h	Reserved	
0h	57h-59h	Reserved	
	60h-68h	Reserved	
	69h	Reserved	
0h	6Ah-6Bh	Reserved	
0h	6Ch-FFh	Reserved	
1h	00h	3DSTATE_DRAWING_RECTANGLE	Strips & Fans
1h	02h	3DSTATE_SAMPLER_PALETTE_LOAD0	Sampling Engine
1h	03h	Reserved	

Opcode Bits 26:24	Sub Opcode Bits 23:16	Command	Definition Chapter
1h	04h	3DSTATE_CHROMA_KEY	Sampling Engine
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	19h	3DSTATE_BINDING_TABLE_POOL_ALLOC	Resource Streamer
1h	1Ah	3DSTATE_GATHER_POOL_ALLOC	Resource Streamer
1h	1Bh	3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC	Resource Streamer
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	Reserved	3D Pipeline
0h	1h	04h	GPGPU CSR BASE ADDRESS	Graphics Processing Engine
0h	1h	05h-1Dh	Reserved	
0h	1h	1Eh	Reserved	
0h	1h	1Fh-20h	Reserved	
0h	1h	21h-24h	Reserved	
0h	1h	25h-FFh	Reserved	N/A
Reserved	**Bits 26:24**	**Bits 23:16**		
0h	2h-7h	XX	Reserved	N/A

VEBOX Commands

The VEBOX commands are used to program the Video Enhancement engine attached to the Video Enhancement Command Parser.

VEBOX Command Map

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command
2h	4h	0h	0h	VEBOX_SURFACE_STATE
2h	4h	0h	2h	VEBOX_STATE
2h	4h	0h	3h	VEBOX_DI_IECP
2h	4h	0h	1h	VEBOX_TILING_CONVERT

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)	SubopA (23:21)	SubopB (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

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

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
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

Scheduling

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

Command Stream Virtual Memory Control

Per-Process GTT (PPGTT) is setup for an engine (Render, Blitter, Video and Video Enhancement) by programming corresponding Page Directory Pointer (PDP) registers listed below. Refer “Graphics Translation Tables” in “Memory Overview” for more details on Per-Process page table entries and related translations.

Execlists

Execlists are the method by which new contexts are submitted for execution. Note that this mechanism cannot be used when the **Execlist Enable** bit in the corresponding engines MODE register is not set, i.e GFX_MODE register for Render Engine, BLT_MODE register for Blitter Engine, VCS_MODE register for Video Engine, or VECS_MODE register for Video Enhancement Engine. If this bit is not set in the engine's MODE register, writing to the registers in this section is UNDEFINED.

Broadwell implements two execlists. Each execlist can have up to two context descriptors in it, each describing a context to run. SW assembles an execlist by writing each of the context descriptor elements to the Execlist Submit Port register. Writing the final DWord triggers the submission. It is the responsibility of SW to keep track of when an empty execlist entry is available to receive a new execlist submitted via the Submit Port. Submitting a new execlist when there is already a pending execlist (in addition to the current execlist) is UNDEFINED. In general, the interrupt indicating that the pending execlist has become the current execlist should always be observed before a new pending execlist is submitted. This includes the case where the ring is idle and the very first execlist is submitted; it should not be assumed that this execlist becomes the current list instantaneously.

The submission of a new execlist (known as a preemption request) is interpreted as a request to switch execlists as soon as possible. This is the only trigger for an execlist switch. Within an execlist, a switch from one element (context) to the next can be triggered for several reasons, all of which are synchronous to what the running context itself is doing. Once a context is switched out, the relevant context state and context descriptor doesn't exist in HW, only way the context can be brought back in to HW is by SW resubmitting the context through Execlist Submit Port.

SW must ensure the contexts submitted to both the context descriptors in the execlist are different; i.e SW must not submit the same context descriptor to both the elements of the execlist.

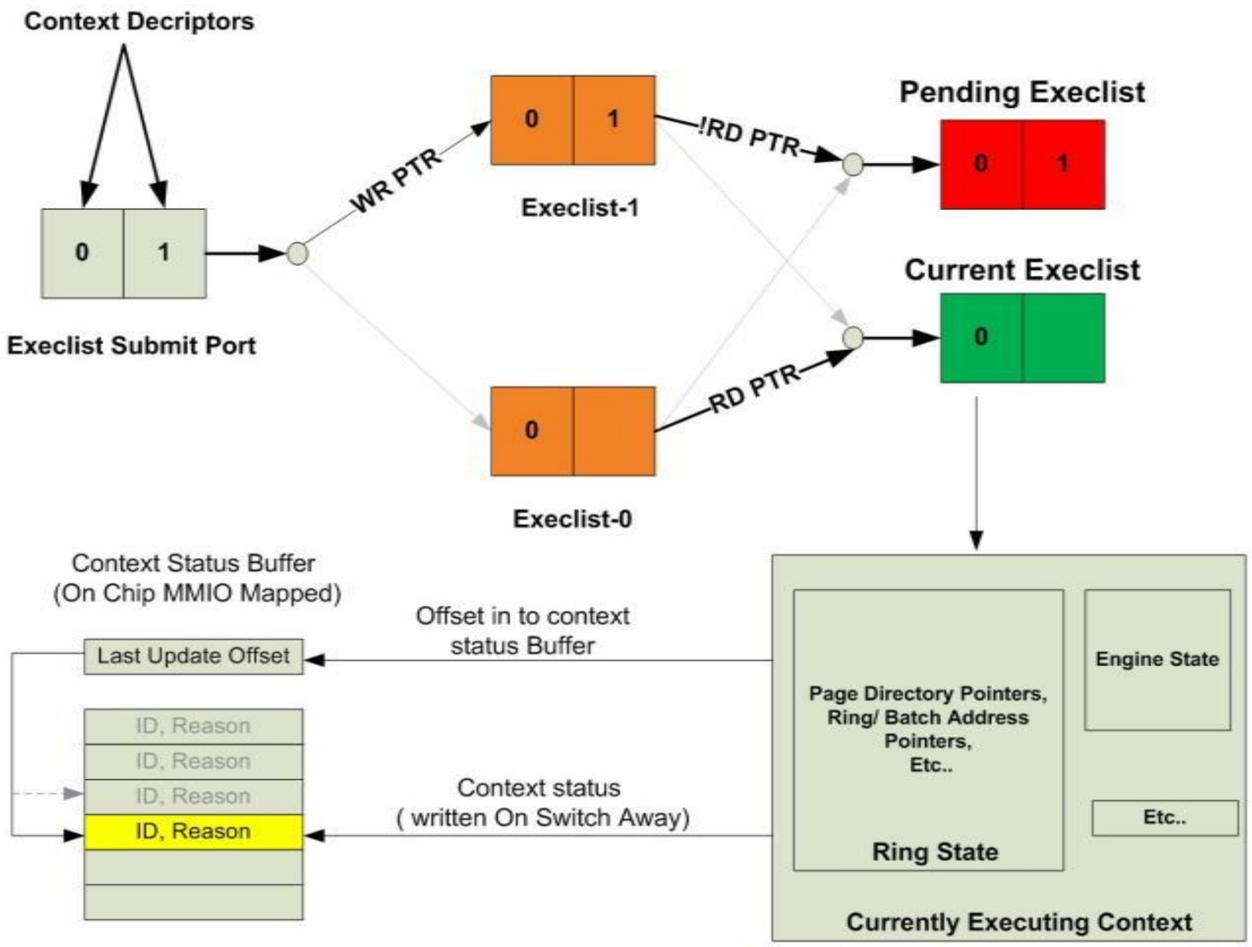
The following are Execlist Registers:

Execlist Submit Port Register

Execlist 1 Contents

EXECLIST_STATUS - Execlist Status Register

Execlist Structure



Before submitting a context for the first time, the context image must be properly initialized. Proper initialization includes the ring context registers (ring location, head/tail pointers, etc.) and the page directory.

Render CS Only: Render state need not be initialized; the **Render Context Restore Inhibit** bit in the Context/Save image in memory should be set to prevent restoring garbage render context. See the Logical Ring Context Format section for details.

Context Descriptor Format Structure

Overall Context Layout

Context Layout

When Execlists are enabled, the Context Image for the rendering engine consists of 20 4K pages:

Per-Process HW Status Page (4K)
Register State Context

When Execlists are disabled, the context image doesn’t consist the Per-Process HW status page.

Register State context is explained in detail in “Register State Context” Section.

Ring Context

Ring Context starts at 4K offset from LRCA. Ring context contains all the details that are needed to be initialized by SW for submitting a context to HW for execution (Ring Buffer Details, Page Directory Information, etc.). Ring context is five cachelines in size. Note that the last cacheline of the ring context is specific for a given Engine and hence SW needs to populate it accordingly.

Ring Context comprises of the EXECLIST CONTEXT, EXECLIST CONTEXT (PPGTT Base) of the register state context. In Ring Buffer mode of scheduling EXECLIST CONTEXT contents are save/restored as NOOPS by HW.

EXECLIST CONTEXT
EXECLIST CONTEXT(PPGTT Base)

Ring Buffer

Ring Buffer can exist anywhere in memory mapped via Global GTT. Ring buffer details are mentioned in the ring context area of LRCA (Ring Buffer - Start Address, Head Offset, Tail Pointer & Control Register) in Execution List mode of scheduling. Ring Buffer registers are directly programmed in Ring Buffer mode of scheduling.

Context Descriptor Format

Context Descriptor Format

Before submitting a context for the first time, the context image must be properly initialized. Proper initialization includes the ring context registers (ring location, head/tail pointers, etc.) and the page directory.

Render CS Only: Render state need not be initialized; the **Render Context Restore Inhibit** bit in the Context/Save image in memory should be set to prevent restoring garbage render context. See the Logical Ring Context Format section for details.

Programming Note on Context ID field in the Context Descriptor

This section describes the current usage by SW.

Layout:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Eng. ID				**SW Counter**					**HW Use**		**SW Context ID**																				

Logical Ring Context Format

Context descriptor has the graphics virtual address pointing to the logical context in memory. Logical context has all the details required for an engine to execute a context. This is the only means through which software can pass on all the required information to hardware for executing a context. Engine first step on selecting a context for execution is to restore (fetch-context save) the logical context from memory to setup the appropriate state in the hardware. Engine on switching out the context from execution saves (store- context restore) the latest updated state to logical context in memory, the updated state is result of the command buffer execution.

When execution lists are enabled, the Logical Context of each engine (Render, Video, Blitter, Video Enhancement, etc.) primarily consists of the following sections:

- Per-Process HW Status Page (4K)
- Ring Context (Ring Buffer Control Registers, Page Directory Pointers, etc.)
- Engine Context (PipelineState, Non-pipelineState, Statistics, MMIO)

Per-Process of HW status Page (PPHWSP)

This is a 4KB scratch space memory allocated for each of the context in global address space. First few cachelines are used by the engine for implicit reports like auto-report of head pointer, timestamp statistics associated with a context execution, rest of the space is available for software as scratch space for reporting fences through MI commands. Context descriptor points to the base of Per-Process HW status page. See the PPHWSP format in **PPHWSP_LAYOUT**.

Logical Ring Context

Logical Ring Context starts immediately following the PPHWSP in memory. Logical ring context is five cachelines in size. This is the minimal set of hardware state required to be programmed by SW for setting up memory access and the ring buffer for a context to be executed on an engine. Memory setup is required for appropriate address translation in the memory interface. Ring buffer details the location of the ring buffer in global graphics virtual address space with its corresponding head pointer and the tail pointer. Ring context also has "Context Save/Restore Control Register-CTXT_SR_CTL" which details the engine context save/restore format. Engine first restores the Logical Ring Context and upon processing CTXT_SR_CTL it further decides the due course of Engine Context restore. Logical Ring Context is mostly identical across all engines. Logical ring context is saved to memory with the latest up to date state when a context is switched out.

Engine Context

Engine context starts immediately following the logical ring context in memory. This state is very specific to an engine and differs from engine to engine. This part of the context consists of the state from all the units in the engine that needs to be save/restored across context switches. Engine restores the engine context following the logical ring context restore. It is tedious for software to populate the engine context as per the requirements, it is recommended to implicitly use engine to populate this portion of the context. Below method can be followed to achieve the same:

- When a context is submitted for the first time for execution, SW can inhibit engine from restoring engine context by setting the "Engine Context Restore Inhibit" bit in CTXT_SR_CTL register of the logical ring context. This will avoid software from populating the Engine Context. Software must program all the state required to initialize the engine in the ring buffer which would initialize the hardware state. On a subsequent context save engine will populate the engine context with appropriate values.
- Above method can be used to create a complete logical context with engine context populated by the hardware. This Logical context can be used as an Golden Context Image or template for subsequently created contexts.

Engine saves the engine context following the logical ring context on switching out a context.

The detailed format of the logical ring context (Blitter/Video/VideoEnhancement) is documented in the Memory Data Formats chapter.

The detailed formats of the Render Logical Ring and Engine Context, including their size, is mentioned in the "Register/State Context" topic for each product.

Context Status

A context switch interrupt will be sent anytime a context switch or execlist change occurs (including the execlist change without context switch scenario described in the ELSP -- Execlist Submit Port Register section). A status QW for the context that was just switched away from will be written to the Context Status Buffer in the Global Hardware Status Page. A copy of the Context Status Buffer is also maintained ON CHIP inside the command streamer, which is MMIO mapped and can be read/written using MMIO access.

Context Status Buffer in Global Hardware Status Page is exercised when IA based scheduling is done. The status contains the context ID and the reason for the context switch. Note that since there will have been no running contexts when the very first (after reset) execlist is submitted or when HW is idle, the Context ID in the first Context Status Qword will be UNDEFINED, this is indicated by setting IDLE to ACTIVE bit in the context status.

Format of Context Status QWord

Bits	Description
63:32	**Context ID**
31:29	Reserved
28	Reserved
27:24	Reserved
23:20	Reserved
19:16	**Display Plane.** This indicates the display plane for which Wait on Scanline/V-Blank/Sync Flip has been executed leading to context switch. This field is only valid when one of the "Wait on Scanline" or "Wait on Vblnak" or "Wait on sync Flip" is set. 0000b: Reserved (Look at field 14:12) 0001b: Reserved 0010b: Reserved 0011b: Display Plane-7 0100b: Display Plane-8 0101b: Display Plane-9 0110b: Display Plane-10 0111b: Display Plane-11 1000b: Display Plane-12 1001b to 1111b: Reserved
15	**Lite Restore.** This bit is only valid only when Preempted bit is set. When set, this bit indicates a given context got preempted with the same context resulting in Lite Restore in HW.

Bits	Description
14:12	**Display Plane.** This indicates the display plane for which Wait on Scanline/V-Blank/Sync Flip has been executed leading to context switch. This field must be looked at only when Display Plane on bits [19:16] is "0000b" and when one of the "Wait on Scanline" or "Wait on Vblank" or "Wait on sync Flip" is set. 000b: Display Plane-1(Pipe A if Wait on V-blank) 001b: Display Plane-2(Pipe B if Wait on V-blank) 010b: Display Plane-3(Pipe C if Wait on V-blank) 011b: Display Plane-4 100b: Display Plane-5 101b: Display Plane-6
11	**Semaphore Wait Mode** 0: Signal Mode 1: Poll Mode This field is valid and must be looked at only when the "Wait on Semaphore" field is set.
10	Reserved
9	Reserved
8	**Wait on Scanline** has resulted in context switch.
7	**Wait on Semaphore** has resulted in context switch.
6	**Wait on V-Blank** has resulted in context switch.
5	**Wait on Sync Flip** has resulted in context switch.
4	**Context Complete** Element is completely processed (Head eqv to Tail) and resulted in a context switch.
3	**ACTIVE to IDLE** following this context switch there is no active element available in HW to execute.
2	**Element Switch.** Context Switch happened from first element in the current execlist to the second element of the same execlist.
1	**Preempted.** Submission of a new execlist has resulted in context switch. The switch is from element in current execlist to element in pending execlist.
0	**IDLE to ACTIVE.** Execlist Submitted when HW is IDLE. When this bit is set rest of the fields in CSQ are not valid.

Context Status should be inferred as described in the tables below. In the two tables below only one of the context switch types will be set and it's quite possible multiple context switch reasons are set. A "Y" in a cell indicates the possibility of the context switch type for the corresponding context switch reason.

Inference of Context Status

Ctx Switch Type Ctx Switch Reason	IDLE to Active	Preempted/ Execlist Switch	**Element Switch	ACTIVE to IDLE
Context Complete	X	Y	Y	Y
Wait on Sync Flip	X	Y	Y	Y
Wait on V-Blank	X	Y	Y	Y
Wait on ScanLine	X	Y	Y	Y
Wait on Semaphore	X	Y	Y	Y

Ctx Switch Type Ctx Switch Reason	IDLE to Active	Preempted/ Execlist Switch	**Element Switch	ACTIVE to IDLE
High Priority Context	Y	Y	X	Y

** This field is not valid when High Priority Context field is set and HW must force it to '0'.

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 to the **Context Status Buffer Write Pointer,** which is recorded in the Context Status Buffer Pointer register. At the end of the context switch interrupt processing SW will update the **Context Status Buffer Read Pointer** with the write buffer pointer value. The status QWs can be examined to determine which contexts were switched out between context interrupt service intervals, and why.

Number of Context Status Entries

Number of Status Entries
6 (QW) Entries

Status QWords are written to the Context Status Buffer at incrementing locations. The Context Status Buffer has a limited size (see Table Number of Context Status Entries) and simply wraps around to the beginning when the end is reached. Normally the number of status updates that can occur without SW intervening to submit a new execlist (and presumably reading any new status) is the number of execlists times the maximum number of context elements per execlist. Also note that there is no predictable relationship between a context's position in an execlist and the position of its corresponding status QWord in the Context Status Buffer.

The Context Status Buffer fits into a single cacheline so that the whole buffer is read from memory at once if the driver performs a cacheable read.

Format of the Context Status Buffer

QW	Description
7	**Last Written Status Offset.** The lower byte of this QWord is written on every context switch with the (pre-increment) value of the b>Context Status Buffer Write Pointer. The lower 3 bits increment for every status QWord write; bits[7:3] are reserved and must be '0'. The lowest 3 bits indicate which of the Context Status QWords was just written. The rest of the bits [63:8] are reserved.
6	Reserved: MBZ
5:0	**Context Status QWords.** A circular buffer of context status QWs. As each context is switched away from, its status is written here at ascending QWs as indicated by the **Last Written Status Offset**. Once QW 5 has been written, the pointer wraps around so that the next status will be written at QW0. Format = ContextStatusDW

The following are Context Status Registers:

CTXT_ST_BUF - Context Status Buffer Contents

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.

Logic Functions Included
• 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. • DMA action for fetching of execlists from memory. • Handling of ring context switch interrupt.

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.

Batch Buffer Privilege Register

FORCE_TO_NONPRIV - FORCE_TO_NONPRIV

Mode Registers

The following are the Mode Registers:

Register
INSTPM - Instruction Parser Mode Register
EXCC - Execute Condition Code Register
NOPID - NOP Identification Register
CSPREEMPT - CSPREEMPT
RS_PRE_HINT - RS Preemption Hint
RS_PREEMPTION_HINT_UDW - RS Preemption Hint UDW
IDLEDLY - Idle Switch Delay
SEMA_WAIT_POLL - Semaphore Polling Interval on Wait
HWS_PGA - Hardware Status Page Address Register
Hardware Status Page Layout

Logical Context Support

The following are the Logical Context Support Registers:

Register
BB_ADDR - Batch Buffer Head Pointer Register
BB_ADDR_UDW - Batch Buffer Upper Head Pointer Register
CCID - Current Context Register
CXT_SIZE - Context Sizes
RS_CXT_OFFSET - Resource Streamer Context Offset
URB_CXT_OFFSET - URB Context Offset
VF_CXT_OFFSET - Vertex Fetch Context Offset
SYNC_FLIP_STATUS - Wait For Event and Display Flip Flags Register
SYNC_FLIP_STATUS_1 - Wait For Event and Display Flip Flags Register 1
SYNC_FLIP_STATUS_2 - Wait For Event and Display Flip Flags Register 2
SBB_ADDR - Second Level Batch Buffer Head Pointer Register
SBB_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Register
SBB_STATE - Second Level Batch Buffer State Register
PS_INVOCATION_COUNT_SLICE0 - PS Invocation Count for Slice0
PS_INVOCATION_COUNT_SLICE1 - PS Invocation Count for Slice1
PS_INVOCATION_COUNT_SLICE2 - PS Invocation Count for Slice2
PS_DEPTH_COUNT_SLICE0 - PS Depth Count for Slice0
PS_DEPTH_COUNT_SLICE1 - PS Depth Count for Slice1
PS_DEPTH_COUNT_SLICE2 - PS Depth Count for Slice2

Register
DISPLAY_MESSAGE_FORWARD_STATUS - Display Message Forward Status Register
R_PWR_CLK_STATE - Render Power Clock State Register
CS_CONTEXT_STATUS1 - Context Status1 for RCS-BE

Context Save Registers

The following are the Context Save Registers:

Register
BB_PREEMPT_ADDR - Batch Buffer Head Pointer Preemption Register
BB_PREEMPT_ADDR_UDW - Batch Buffer Upper Head Pointer Preemption Register
RING_BUFFER_HEAD_PREEMPT_REG - RING_BUFFER_HEAD_PREEMPT_REG
BB_START_ADDR - Batch Buffer Start Head Pointer Register
BB_START_ADDR_UDW - Batch Buffer Start Upper Head Pointer Register
BB_ADDR_DIFF - Batch Address Difference Register
BB_OFFSET - Batch Offset Register
SBB_PREEMPT_ADDR - Second Level Batch Buffer Head Pointer Preemption Register
SBB_PREEMPT_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Preemption Register
BB_PER_CTX_PTR - Batch Buffer Per Context Pointer
GAM_CTX - GAM Context Save Register
LNCF_CTX - LNCF Context Save Register
TDL_CONTEXT_SAVE - Context Save Request to TDL

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.

The commands detailed in this chapter are used across product families. 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.

Commands
MI_NOOP
MI_ARB_CHECK
MI_ARB_ON_OFF
MI_BATCH_BUFFER_START
MI_BATCH_BUFFER_END
MI_CONDITIONAL_BATCH_BUFFER_END

Commands
MI_DISPLAY_FLIP
MI_LOAD_SCAN_LINES_EXCL
MI_LOAD_SCAN_LINES_INCL
MI_CLFLUSH
MI_MATH
MI_REPORT_HEAD
MI_STORE_DATA_IMM
MI_STORE_DATA_INDEX
MI_ATOMIC
MI_COPY_MEM_MEM
MI_LOAD_REGISTER_REG
MI_LOAD_REGISTER_MEM
MI_STORE_REGISTER_MEM
MI_SUSPEND_FLUSH
MI_USER_INTERRUPT
MI_WAIT_FOR_EVENT
MI_SEMAPHORE_SIGNAL
MI_SEMAPHORE_WAIT
MI_FORCE_WAKEUP

Watchdog Timer Registers

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

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

Programming Notes: When watch dog timer is enabled, HW does not trigger any kind of idle sequences. SW must enable and disable watch dog timer for any given workload within the same command buffer dispatch. SW must disable watch dog timer around semaphore waits and wait for events commands so that HW can trigger appropriate idle sequence for power savings.

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:

Bit Definition for Interrupt Control Registers - Render

Bit Definition for Interrupt Control Registers - Blitter

Bit Definition for Interrupt Control Registers - Media#1

Bit Definition for Interrupt Control Registers - Media#2

Bit Definition for Interrupt Control Registers - Video Enhancement

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	Reserved	
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. This interrupt is for handling Legacy Page Fault interface for all Command Streamers (BCS, RCS, VCS, VECS). When Fault Repair Mode is enabled, Interrupt mask register value is not looked at to generate interrupt due to page fault. Please refer to vol1c "Page Fault Support" section for more details.	Set when event occurs, cleared when event cleared. Not supported to be unmasked.
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. Only for Media Pipe.
5	**L3 Parity interrupt**	Only for Render Pipe
4	**Flush Notify Enable**	0
3	**Master Error**	Set when error occurs, cleared when error cleared.
2	Reserved	
0	**User Interrupt**	0

IMR - Interrupt Mask Register

Command Streamer (All) > Hardware Status 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 structures describe the Hardware-Detected Error bits:

RCS Hardware-Detected Error Bit Definitions Structure

BCS Hardware-Detected Error Bit Definitions Structure

VCS Hardware-Detected Error Bit Definitions Structure

VECS Hardware-Detected Error Bit Definitions Structure

The following are the EIR, EMR, and ESR registers:

EIR - Error Identity Register

EMR - Error Mask Register

ESR - Error Status Register

Blitter Engine Command Streamer (BCS)

The BCS (Blitter Command Streamer) unit primarily serves as the software programming interface between the O/S driver and the Blitter Engine. It is responsible for fetching, decoding, and dispatching of data packets (Blitter Commands) to the front end interface module of Blitter Engine.

Logic Functions Included
• 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 blit engine. • Handling of user interrupts. • Flushing the Blitter Engine. • Handle NOP. • DMA action for fetching of execlists from memory. • Handling of ring context switch interrupt.

The register programming bus is a DWord interface bus that is driven by the configuration master. The BCS unit only claims memory mapped I/O cycles that are targeted to its range of 0x22000 to 0x224FF. The Blitter, Render and Media Engines use semaphore to synchronize their operations.

BCS operates completely independent of the other render and media command streams.

The simple sequence of events is as follows: a ring (say PRB0) is programmed by a memory-mapped register write cycle. The DMA inside BCS 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 Blit 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.

Logical Context Support

The following are the Logical Context Support Registers:

Register
BB_ADDR - Batch Buffer Head Pointer Register
BB_ADDR_UDW - Batch Buffer Upper Head Pointer Register
SBB_ADDR - Second Level Batch Buffer Head Pointer Register
SBB_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer 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
SYNC_FLIP_STATUS_2 - Wait For Event and Display Flip Flags Register 2
DISPLAY_MESSAGE_FORWARD_STATUS - Display Message Forward Status Register
BB_START_ADDR - Batch Buffer Start Head Pointer Register
BB_START_ADDR_UDW - Batch Buffer Start Upper Head Pointer Register
BB_ADDR_DIFF - Batch Address Difference Register
CCID - Current Context Register
SBB_STATE - Second Level Batch Buffer State Register
BB_OFFSET - Batch Offset Register
RING_BUFFER_HEAD_PREEMPT_REG - RING_BUFFER_HEAD_PREEMPT_REG
BB_PREEMPT_ADDR - Batch Buffer Head Pointer Preemption Register
BB_PREEMPT_ADDR_UDW - Batch Buffer Upper Head Pointer Preemption Register
SBB_PREEMPT_ADDR - Second Level Batch Buffer Head Pointer Preemption Register
SBB_PREEMPT_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Preemption Register
MI_PREDICATE_RESULT_1 - Predicate Rendering Data Result 1
MI_PREDICATE_RESULT_2 - Predicate Rendering Data Result 2
INDIRECT_CTX - Indirect Context Pointer
INDIRECT_CTX_OFFSET - Indirect Context Offset Pointer
BB_PER_CTX_PTR - Batch Buffer Per Context Pointer

Mode Registers

The following are Mode Registers:

Registers
BCS_CXT_SIZE - BCS Context Sizes
MI_MODE - Mode Register for Software Interface

Mode Registers (continued)

Reisters
INSTPM - Instruction Parser Mode Register
EXCC - Execute Condition Code Register
IDLEDLY - Idle Switch Delay
SEMA_WAIT_POLL - Semaphore Polling Interval on Wait
RESET_CTRL - Reset Control Register
PREEMPTION_HINT - Preemption Hint
PREEMPTION_HINT_UDW - Preemption Hint Upper DWord

Register
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. 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.

Commands
MI_NOOP
MI_ARB_CHECK
MI_ARB_ON_OFF
MI_BATCH_BUFFER_START

The following table lists the non-privileged registers that can be written to from a non-secure batch buffer executed from Render Command Streamer.

User Mode Non-Privileged Registers

MMIO Name	MMIO Offset	Size in DWords
BCS_GPR	22600h	32
BCS_SWCTRL	22200h	32

Commands
MI_BATCH_BUFFER_END
MI_CONDITIONAL_BATCH_BUFFER_END
MI_DISPLAY_FLIP
MI_LOAD_SCAN_LINES_EXCL
MI_LOAD_SCAN_LINES_INCL
MI_FLUSH_DW
MI_MATH
MI_REPORT_HEAD
MI_STORE_DATA_IMM
MI_STORE_DATA_INDEX
MI_ATOMIC
MI_COPY_MEM_MEM
MI_LOAD_REGISTER_REG
MI_LOAD_REGISTER_MEM
MI_STORE_REGISTER_MEM
MI_SUSPEND_FLUSH
MI_USER_INTERRUPT
MI_WAIT_FOR_EVENT
MI_SEMAPHORE_SIGNAL
MI_SEMAPHORE_WAIT
MI_FORCE_WAKEUP

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
- DMA action for fetching of execlists and ring data from memory
- Management of the Head pointer for the Ring Buffer

- Decode of ring data and sending it to the appropriate destination: AVC, VC1, or MPEG2 engine
- Handling of user interrupts
- Handling of ring context switch interrupt
- Flushing the MFX Engine
- Handle NOP

The register programming (RM) bus is a DWord interface bus that is driven by the Gx Command Streamer. The VCS unit 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.

Watchdog Timer Registers

The following registers are defined as Watchdog Timer registers:

PR_CTR_CTL - Watchdog Counter Control

PR_CTR_THRSH - Watchdog Counter Threshold

Logical Context Support

This section contains the registers for Logical Context Support.

Register
BB_STATE - Batch Buffer State Register
BB_START_ADDR - Batch Buffer Start Head Pointer Register
BB_START_ADDR_UDW - Batch Buffer Start Upper Head Pointer Register
BB_ADDR_DIFF - Batch Address Difference Register
BB_ADDR - Batch Buffer Head Pointer Register
SBB_ADDR - Second Level Batch Buffer Head Pointer Register
SBB_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Register
CCID - Current Context Register
SBB_STATE - Second Level Batch Buffer State Register
BB_OFFSET - Batch Offset Register
RING_BUFFER_HEAD_PREEMPT_REG - RING_BUFFER_HEAD_PREEMPT_REG
BB_PREEMPT_ADDR - Batch Buffer Head Pointer Preemption Register
BB_PREEMPT_ADDR_UDW - Batch Buffer Upper Head Pointer Preemption Register
SBB_PREEMPT_ADDR - Second Level Batch Buffer Head Pointer Preemption Register
SBB_PREEMPT_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Preemption Register
MI_PREDICATE_RESULT_1 - Predicate Rendering Data Result 1
MI_PREDICATE_RESULT_2 - Predicate Rendering Data Result 2
DISPLAY_MESSAGE_FORWARD_STATUS - Display Message Forward Status Register
FORCE_TO_NONPRIV - FORCE_TO_NONPRIV
INDIRECT_CTX - Indirect Context Pointer
INDIRECT_CTX_OFFSET - Indirect Context Offset Pointer
BB_PER_CTX_PTR - Batch Buffer Per Context Pointer
SYNC_FLIP_STATUS - Wait For Event and Display Flip Flags Register
SYNC_FLIP_STATUS_1 - Wait For Event and Display Flip Flags Register 1
SYNC_FLIP_STATUS_2 - Wait For Event and Display Flip Flags Register 2

Mode Registers

The following are Mode Registers:

Mode Register
MI_MODE - Mode Register for Software Interface
INSTPM - Instruction Parser Mode Register
NOPID - NOP Identification Register
IDLEDLY - Idle Switch Delay
RESET_CTRL - Reset Control Register
PREEMPTION_HINT - Preemption Hint
PREEMPTION_HINT_UDW - Preemption Hint Upper DWord
SEMA_WAIT_POLL - Semaphore Polling Interval on Wait

Misc Register
HWS_PGA - Hardware Status Page Address Register
Hardware Status Page Layout

Registers in Media Engine

This topic 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.

The registers detailed in this chapter are used across multiple projects and are extensions to previous projects. However, slight changes may be present in some registers (i.e., for features added or removed), or some registers may be removed entirely. These changes are clearly marked within this chapter.

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.

TIMESTAMP - Reported Timestamp Count

CTX_TIMESTAMP - Context Timestamp Count

Memory Interface Commands for Video Codec 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 Video Codec Engine.

The commands detailed in this chapter are used across product families. 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 Commands
MI_ARB_CHECK
MI_BATCH_BUFFER_END
MI_CONDITIONAL_BATCH_BUFFER_END
MI_BATCH_BUFFER_START
MI_FLUSH_DW
MI_COPY_MEM_MEM
MI_LOAD_REGISTER_REG
MI_MATH
MI_NOOP
MI_REPORT_HEAD
MI_SEMAPHORE_SIGNAL
MI_SEMAPHORE_WAIT
MI_STORE_REGISTER_MEM
MI_STORE_DATA_IMM
MI_STORE_DATA_INDEX
MI_SUSPEND_FLUSH
MI_USER_INTERRUPT
MI_LOAD_REGISTER_MEM
MI_ATOMIC
MI_FORCE_WAKEUP

Video Enhancement Engine Command Interface

The following topics describe the Video Enhancement Engine Command Interface.

VECS_RINGBUF — Ring Buffer Registers

The following are 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

Logical Context Support

The following are Logical Context Support Registers:

Register
BB_ADDR - Batch Buffer Head Pointer Register
BB_ADDR_UDW - Batch Buffer Upper Head Pointer Register
SBB_ADDR - Second Level Batch Buffer Head Pointer Register
SBB_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Register
BB_STATE - Batch Buffer State Register
TIMESTAMP - Reported Timestamp Count
CTX_TIMESTAMP - Context Timestamp Count
BB_START_ADDR - Batch Buffer Start Head Pointer Register
BB_START_ADDR_UDW - Batch Buffer Start Upper Head Pointer Register
BB_ADDR_DIFF - Batch Address Difference Register
CCID - Current Context Register
SBB_STATE - Second Level Batch Buffer State Register
BB_OFFSET - Batch Offset Register
RING_BUFFER_HEAD_PREEMPT_REG - RING_BUFFER_HEAD_PREEMPT_REG
BB_PREEMPT_ADDR - Batch Buffer Head Pointer Preemption Register
BB_PREEMPT_ADDR_UDW - Batch Buffer Upper Head Pointer Preemption Register
SBB_PREEMPT_ADDR - Second Level Batch Buffer Head Pointer Preemption Register
SBB_PREEMPT_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Preemption Register
MI_PREDICATE_RESULT_1 - Predicate Rendering Data Result 1
MI_PREDICATE_RESULT_2 - Predicate Rendering Data Result 2
DISPLAY_MESSAGE_FORWARD_STATUS - Display Message Forward Status Register
FORCE_TO_NONPRIV - FORCE_TO_NONPRIV
INDIRECT_CTX - Indirect Context Pointer
INDIRECT_CTX_OFFSET - Indirect Context Offset Pointer

Register
BB_PER_CTX_PTR - Batch Buffer Per Context Pointer
SYNC_FLIP_STATUS - Wait For Event and Display Flip Flags Register
SYNC_FLIP_STATUS_1 - Wait For Event and Display Flip Flags Register 1
SYNC_FLIP_STATUS_2 - Wait For Event and Display Flip Flags Register 2

Mode Registers

Register
HWS_PGA - Hardware Status Page Address Register
Hardware Status Page Layout

MI Commands for Video Enhancement 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 Video Codec Engine.

The commands detailed in this chapter are used across product families. 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 Command
MI_ARB_CHECK
MI_BATCH_BUFFER_END
MI_CONDITIONAL_BATCH_BUFFER_END
MI_BATCH_BUFFER_START
MI_FLUSH_DW
MI_LOAD_REGISTER_REG
MI_MATH
MI_NOOP
MI_REPORT_HEAD
MI_SEMAPHORE_SIGNAL
MI_SEMAPHORE_WAIT
MI_ATOMIC
MI_STORE_REGISTER_MEM
MI_STORE_DATA_IMM
MI_STORE_DATA_INDEX
MI_SUSPEND_FLUSH
MI_USER_INTERRUPT
MI_COPY_MEM_MEM
MI_LOAD_REGISTER_MEM
MI_FORCE_WAKEUP

Watchdog Timers

Watchdog Counter Control

The Watchdog Counter Control determines if the watchdog is enabled, disabled and count mode. The watchdog is enabled is when the value of the register [30:0] is equal to zero([30:0] = 'd0). If enabled, then the Watchdog Counter is allowed to increment. The watchdog is disabled is when the value of the register [30:0] is equal to one where only bit zero is a value of '1'([30:0] = 0x00000001). If disabled, then the value of Watchdog Counter is reset to a value of zero. Bit 31, specifies the counting mode. If bit 31 is zero, then we will count based timestamp toggle (refer to Reported Timestamp Count register for toggle time). If bit 31 is one, then we will count every ungated GPU clock.

Programming Notes: When watch dog timer is enabled, HW does not trigger any kind of idle sequences. SW must enable and disable watch dog timer for any given workload within the same command buffer dispatch. SW must disable watch dog timer around semaphore waits and wait for events commands so that HW can trigger appropriate idle sequence for power savings.

This register is context saved as part of engine context.

Watchdog Counter Threshold

If the Watchdog Counter Threshhold is equal to Watchdog Counter, then the interrupt bit is set in the IIR(bit 6) and the Watchdog Counter is reset to zero.

This register is context saved as part of engine context.

Watchdog Counter

The Watchdog Counter is the count value of the watchdog timer. The Counter can be reset due to the Watchdog Counter Control being disabled or being equal to the Watchdog Counter Threshhold. The increment of the Watchdog counter is enabled when the Watchdog Counter Control is enabled and the current context is valid and execlist is enabled which includes the time to execute, flush and save the context.

The increment of the Watchdog counter is under the following conditions:

- Watchdog timer is enabled.
- Context is valid

The increment granularity is based controlled by Watchdog Counter Control mode(bit 31).

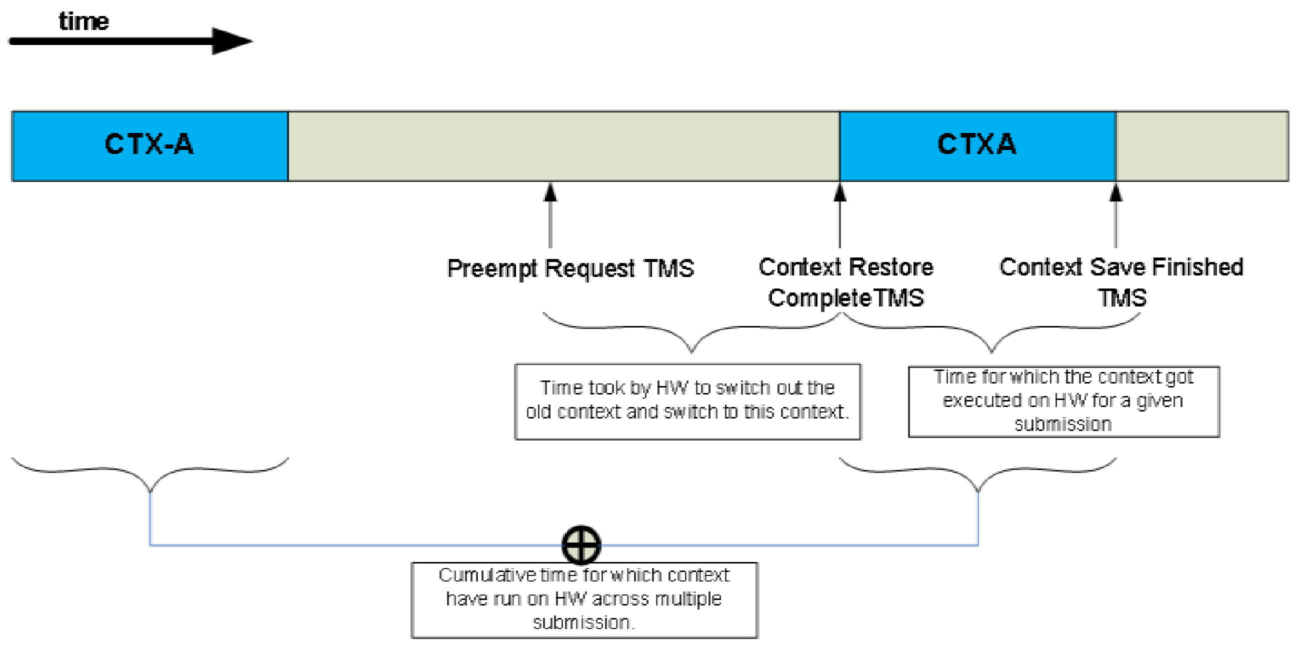
This register is not context saved and restored.

The Per-Process Hardware Status Page

The layout of the Per-Process Hardware Status Page is defined at **PPHWSP_LAYOUT**.

The DWord offset values in the table are in decimal.

Figure below explains the different timestamp values reported to PPHWSP on a context switch.


time
CTX-A
CTXA
Preempt Request TMS
Context Restore CompleteTMS
Context Save Finished TMS
Time took by HW to switch out the old context and switch to this context.
Time for which the context got executed on HW for a given submission
Cumulative time for which context have run on HW across multiple submission.

This page is designed to be read by SW to glean additional details about a context beyond what it can get from the context status.

Accesses to this page are automatically treated as cacheable and snooped. It is therefore illegal to locate this page in any region where snooping is illegal (such as in stolen memory).

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; the 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 its 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_BUFFER_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 commands 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 tables list the Preemptable Commands in the Ring Buffer mode of scheduling:

Engine (below)	**Preemptable Commands**				
	MI_ARB_CHECK	**3DPRIMITIVE**	**GPGPU_WALKER**	**PIPE_CONTROL *****	**MEDIA STATE FLUSH**
Render	AP	Object Level (if enabled *)	Mid-Thread (if enabled **)	PIPESEL-GPGPU MODE	PIPESEL-GPGPU MODE
Blitter	AP	N/A	N/A	N/A	N/A
Media	AP	N/A	N/A	N/A	N/A
VideoEnhancement	AP	N/A	N/A	N/A	N/A

AP: Allow preemption on UHPTR valid and arbitration enabled. Arbitration can be enabled/disabled using MI_ARB_ON_OFF command.

ExecList Scheduling

In ExecList mode of scheduling SW triggers preemption by submitting a new pending execlist to ELSP (ExecList Submit Port). HW triggers preemption on a preemptable command on detecting the availability of the new pending execlist, following preemption context switch happens to the newly submitted execlist. As part of the context switch preempted context state is saved to the preempted context LRCA, context state contains the details such that on resubmission of the preempted context HW can resume execution from the point where it was preempted.

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.
```

The following tables list the Preemptable Commands in ExecList mode of scheduling:

Engine (below)	Preemptable Commands							
	MI_ARB_CHECK	Element Boundary	Semaphore Wait	Wait for Event	3DPRIMITIVE	GPGPU_WALKER	PIPE_CONTROL ***	MEDIA STATE FLUSH
Render	AP	AP	Unsuccessful & AP	Unsuccessful & AP	Object Level (if enabled *)	Mid-Thread (if enabled **)	PIPESEL-GPGPU MODE	PIPESEL-GPGPU MODE
Blitter	AP	AP	Unsuccessful & AP	Unsuccessful & AP	N/A	N/A	N/A	N/A
Media	AP	AP	Unsuccessful & AP	N/A	N/A	N/A	N/A	N/A
Video Enhancement	AP	AP	Unsuccessful & AP	N/A	N/A	N/A	N/A	N/A

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.

Table Notes:

AP - Allow Preemption if arbitration is enabled.

* 0x229c bit 11 determines whether the level of preemption is command or object level.

** 0x20E4 bits 2:1 determine the level of preemption for GPGPU workloads.

*** MI_ATOMIC and MI_SEMAPHORE_SIGNAL commands with Post Sync Op bit set are treated as PIPE_CONTROL command with Post Sync Operation as Atomics or Semaphore Signal.

Command Streamer (CS) ALU Programming

The command streamer implements a rudimentary Arithmetic Logic Unit (ALU) which supports basic arithmetic (Addition and Subtraction) and logical operations (AND, OR, XOR) on two 64-bit operands.

The ALU has two 64-bit registers at the input, SRCA and SRCB, to which source operands are loaded. The ALU result is written to a 64-bit accumulator. The Zero Flag and Carry Flag are assigned based on the accumulator output.

See the ALU Programming section in the Render Engine Command Streamer, for a description of the ALU programming model. That model is the same for all command streamers that support ALU programming, but each command streamer uses different address offsets for the registers used. The following subsections describe the ALU registers in the Blitter command streamer.

CS ALU Programming and Design

MI Commands for Graphics Processing Engines

This chapter lists the MI Commands that are supported by Generic Command Streamer Front End implemented in the graphics processing engines (Render, Video, Blitter and Video Enhance.

Command
MI_NOOP
MI_ARB_CHECK
MI_BATCH_BUFFER_START
MI_BATCH_BUFFER_END
MI_CONDITIONAL_BATCH_BUFFER_END
MI_DISPLAY_FLIP
MI_LOAD_SCAN_LINES_EXCL
MI_LOAD_SCAN_LINES_INCL
MI_CLFLUSH
MI_MATH
MI_REPORT_HEAD
MI_STORE_DATA_IMM
MI_STORE_DATA_INDEX
MI_ATOMIC
MI_COPY_MEM_MEM
MI_LOAD_REGISTER_REG
MI_LOAD_REGISTER_MEM
MI_STORE_REGISTER_MEM
MI_USER_INTERRUPT
MI_WAIT_FOR_EVENT
MI_SEMAPHORE_SIGNAL
MI_SEMAPHORE_WAIT

User Mode Privileged Commands

A subset of the commands are privileged. These commands may be issued only from a privileged batch buffer or directly from a ring. Batch buffers in GGTT memory space are privileged and batch buffers in PPGTT memory space are non-privileged. On parsing privileged command from a non-privileged batch buffer, a Command Privilege Violation Error is flagged and the command is dropped. Command Privilege Violation Error is logged in Error identity register of command streamer which gets propagated as “Command Parser Master Error” interrupt to SW. Privilege access violation checks in HW can be disabled by setting “Privilege Check Disable” bit in GFX_MODE register. When privilege access checks are disabled HW executes the Privilege command as expected.

User Mode Privileged Commands

User Mode Privileged Command	Function in Non-Privileged Batch Buffers	Source
MI_UPDATE_GTT	Command is converted to NOOP.	*CS
MI_STORE_DATA_IMM	Command is converted to NOOP if **Use Global GTT** is enabled.	*CS
MI_STORE_DATA_INDEX	Command is converted to NOOP.	*CS
MI_STORE_REGISTER_MEM	Register read is always performed. Memory update is dropped if **Use Global GTT** is enabled.	*CS
MI_BATCH_BUFFER_START	Command when executed from a batch buffer can set its “Privileged” level to its parent batch buffer or lower. Chained or Second level batch buffer can be “Privileged” only if the parent or the initial batch buffer is “Privileged”. This is HW enforced.	*CS
MI_LOAD_REGISTER_IMM	Command is converted to NOOP if the register accessed is privileged.	*CS
MI_LOAD_REGISTER_MEM	Command is converted to NOOP if **Use Global GTT** is enabled. Command is converted to NOOP if the register accessed is privileged.	*CS
MI_LOAD_REGISTER_REG	Register write to a **Privileged Register** is discarded.	*CS
MI_REPORT_PERF_COUNT	Command is converted to NOOP if **Use Global GTT** is enabled.	Render CS
PIPE_CONTROL	Still send flush down, Post-Sync Operation is NOOP if **Use Global GTT** or Use “Store Data Index” is enabled. Post-Sync Operation LRI to **Privileged Register** is discarded.	Render CS
MI_SET_CONTEXT	Command is converted to NOOP.	Render CS
MI_ATOMIC	Command is converted to NOOP if **Use Global GTT** is	Render CS

User Mode Privileged Command	Function in Non-Privileged Batch Buffers	Source
	enabled.	
MI_COPY_MEM_MEM	Command is converted to NOOP if **Use Global GTT** is used for source or destination address.	*CS
MI_SEMAPHORE_WAIT	Command is converted to NOOP if **Use Global GTT** is enabled.	*CS
MI_ARB_ON_OFF	Command is converted to NOOP.	*CS
MI_DISPLAY_FLIP	Command is converted to NOOP.	*CS
MI_CONDITIONAL_BATCH_BUFFER_END	Command is converted to NOOP if **Use Global GTT** is enabled.	*CS
MI_FLUSH_DW	Still send flush down, Post-Sync Operation is converted to NOOP if **Use Global GTT** or Use "Store Data Index" is enabled.	Blitter CS, Video CS, Video Enhancement CS

Parsing one of the commands in the table above from a non-privileged batch buffer flags an error and converts the command to a NOOP.

The tables below list the non-privileged registers that can be written to from a non-privileged batch buffer executed from various command streamers.

User Mode Non-Privileged Registers for Render Command Streamer (RCS)

MMIO Name	MMIO Offset	Size in DWords
Cache_Mode_0	0x7000	1
Cache_Mode_1	0x7004	1
GT_MODE	0x7008	1
L3_Config	0x7034	1
TD_CTL	0xE400	1
TD_CTL2	0xE404	1
L3SQCREG4	0xB118	1
NOPID	0x2094	1
INSTPM	0x20C0	1
IA_VERTICES_COUNT	0x2310	2
IA_PRIMITIVES_COUNT	0x2318	2
VS_INVOCATION_COUNT	0x2320	2
HS_INVOCATION_COUNT	0x2300	2
DS_INVOCATION_COUNT	0x2308	2
GS_INVOCATION_COUNT	0x2328	2
GS_PRIMITIVES_COUNT	0x2330	2
SO_NUM_PRIMS_WRITTEN0	0x5200	2

MMIO Name	MMIO Offset	Size in DWords
SO_NUM_PRIMS_WRITTEN1	0x5208	2
SO_NUM_PRIMS_WRITTEN2	0x5210	2
SO_NUM_PRIMS_WRITTEN3	0x5218	2
SO_PRIM_STORAGE_NEEDED0	0x5240	2
SO_PRIM_STORAGE_NEEDED1	0x5248	2
SO_PRIM_STORAGE_NEEDED2	0x5250	2
SO_PRIM_STORAGE_NEEDED3	0x5258	2
SO_WRITE_OFFSET0	0x5280	1
SO_WRITE_OFFSET1	0x5284	1
SO_WRITE_OFFSET2	0x5288	1
SO_WRITE_OFFSET3	0x528C	1
CL_INVOCATION_COUNT	0x2338	2
CL_PRIMITIVES_COUNT	0x2340	2
PS_INVOCATION_COUNT_0	0x22C8	2
PS_DEPTH_COUNT _0	0x22D8	2
PS_INVOCATION_COUNT_1	0x22F0	2
PS_DEPTH_COUNT _1	0x22F8	2
PS_INVOCATION_COUNT_2	0x2448	2
PS_DEPTH_COUNT_2	0x2450	2
GPUGPU_DISPATCHDIMX	0x2500	1
GPUGPU_DISPATCHDIMY	0x2504	1
GPUGPU_DISPATCHDIMZ	0x2508	1
MI_PREDICATE_SRC0	0x2400	1
MI_PREDICATE_SRC0	0x2404	1
MI_PREDICATE_SRC1	0x2408	1
MI_PREDICATE_SRC1	0x240C	1
MI_PREDICATE_DATA	0x2410	1
MI_PREDICATE_DATA	0x2414	1
MI_PREDICATE_RESULT	0x2418	1
MI_PREDICATE_RESULT_1	0x241C	1
MI_PREDICATE_RESULT_2	0x23BC	1
3DPRIM_END_OFFSET	0x2420	1
3DPRIM_START_VERTEX	0x2430	1
3DPRIM_VERTEX_COUNT	0x2434	1
3DPRIM_INSTANCE_COUNT	0x2438	1
3DPRIM_START_INSTANCE	0x243C	1
3DPRIM_BASE_VERTEX	0x2440	1

MMIO Name	MMIO Offset	Size in DWords
GPGPU_THREADS_DISPATCHED	0x2290	2
BB_OFFSET	0x2158	1
CS_GPR (1-16)	0x2600	32
OA_CTX_CONTROL	0x2360	1
OACTXID	0x2364	1
OA CONTROL	0x2B00	1
PERF_CNT_1_DW0	0x91b8	1
PERF_CNT_1_DW1	0x91bc	1
PERF_CNT_2_DW0	0x91c0	1
PERF_CNT_2_DW1	0x91c4	1

User Mode Non-Privileged Registers for Blitter Command Streamer (BCS)

MMIO Name	MMIO Offset	Size in DWords
BCS_GPR	0x22600	32
BCS_SWCTRL	0x22200	1

This table represents the Base offset for Video Command Streamers and Media Engine message range:

Unit	MMIO Base Offset	Description
VCS/MFC	0x13000	Video Command Streamer 0
VCS1/MFC1	0x1D000	Video Command Streamer 1
VECS	0x1B000	Video Enhancement Command Streamer
HEVC	0x1E900	

User Mode Non-Privileged Registers for Video Enhancement Command Streamer (VECS)

MMIO Name	MMIO Offset	Size in DWords
VECS_GPR	0x600	32

User Mode Non-Privileged Registers for Video Command Streamer (ALL VCS)

MMIO Name	MMIO Range	Size in DWords
VCS_GPR	0x600	32
MFC_VDBOX1	0x800-0xFFF	512
HEVC	0x00	64
HEVC-Enc	0x00	64