

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

Volume 6: Command Stream Programming

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform
(Cherryview/Braswell graphics)

June 2015, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

| | |
|--|-----------|
| Graphics Command Formats..... | 1 |
| Command Header..... | 2 |
| Memory Interface Commands..... | 6 |
| 2D Commands..... | 8 |
| 3D Commands..... | 10 |
| VEBOX Commands..... | 14 |
| MFX Commands..... | 15 |
| Scheduling..... | 17 |
| Execlists..... | 17 |
| Context Descriptor Format..... | 19 |
| Logical Ring Context Format..... | 20 |
| Context Status..... | 21 |
| RINGBUF — Ring Buffer Registers..... | 25 |
| Virtual Memory Control..... | 25 |
| Render Engine Command Streamer (RCS)..... | 26 |
| Batch Buffer Privilege Register..... | 27 |
| Mode Registers..... | 27 |
| Logical Context Support..... | 28 |
| Context Save Registers..... | 29 |
| MI Commands for Render Engine..... | 30 |
| User Mode Privileged Commands..... | 31 |
| Watchdog Timer Registers..... | 34 |
| Command Stream Front End..... | 35 |
| Interrupt Control Registers..... | 35 |
| Hardware-Detected Error Bit Definitions (for EIR EMR ESR)..... | 37 |
| Blitter Engine Command Streamer (BCS)..... | 38 |
| Watchdog Timer Registers..... | 39 |
| Hardware-Detected Error Bit Definitions (for EIR EMR ESR)..... | 39 |
| Logical Context Support..... | 40 |
| Mode Registers..... | 41 |
| MI Commands for Blitter Engine..... | 42 |

| | |
|---|-----------|
| User Mode Privileged Commands | 45 |
| Video Command Streamer (VCS) | 46 |
| Watchdog Timer Registers | 47 |
| VCS Hardware - Detected Error Bit Definitions (for EIR EMR ESR) | 47 |
| Logical Context Support | 48 |
| Mode Registers | 49 |
| Registers in Media Engine | 49 |
| GFX Pending TLB Cycles Information Registers | 49 |
| Memory Interface Commands for Video Codec Engine | 50 |
| User Mode Privileged Commands | 51 |
| Video Enhancement Engine Command Interface | 53 |
| VECS_RINGBUF — Ring Buffer Registers | 53 |
| Watchdog Timer Registers | 53 |
| Logical Context Support | 54 |
| Mode Registers | 55 |
| MI Commands for Video Enhancement Engine | 56 |
| User Mode Privileged Commands | 57 |
| Preemption | 58 |
| Ring Buffer Scheduling | 58 |
| ExecList Scheduling | 60 |
| Command Streamer (CS) ALU Programming | 61 |

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 |

| Bits | | | | | | |
|-------------|-------|-------|--------------------|----|----|---------------------|
| TYPE | 31:29 | 28:24 | | 23 | 22 | 21:0 |
| 3DPrimitive | 011 | 11 | Opcode – 011 | | | Data DWord Count |
| Reserved | 011 | 11 | Opcode – 100 – 111 | | | |
| Reserved | 100 | XX | | | | |
| Reserved | 101 | XX | | | | |
| Reserved | 110 | XX | | | | |

Notes:

¹The 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 | |
| 15h | Reserved | |
| 16h | MI_SEMAPHORE_MBOX | All |
| 17h | Reserved | |
| 18h | MI_SET_CONTEXT | Render |
| 19h | MI_URB_CLEAR | Render |
| 1Ah | MI_MATH | All |
| 1Bh | MI_SEMAPHORE_SIGNAL | All |

| Opcode (28:23) | Command | Pipes |
|--------------------------|---------------------------------|-------------------|
| 1 DWord | | |
| 1Ch | MI_SEMAPHORE_WAIT | All |
| 1Eh-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 |

| Opcode (28:22) | Command |
|-------------------|--|
| 71h | XY_MONO_SRC_COPY_IMMEDIATE_BLT |
| 72h | XY_PAT_BLT_IMMEDIATE |
| 73h | XY_SRC_COPY_CHROMA_BLT |
| 74h | XY_FULL_IMMEDIATE_PATTERN_BLT |
| 75h | XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT |
| 76h | XY_PAT_CHROMA_BLT |
| 77h | XY_PAT_CHROMA_BLT_IMMEDIATE |
| 78h-7Fh | Reserved |

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 | 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 |
| 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 |

| Opcode Bits 26:24 | Sub Opcode Bits 23:16 | Command | Definition Chapter |
|----------------------|--------------------------|---|--------------------|
| 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 |
| 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 |
| 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 |

| Opcode Bits 26:24 | Sub Opcode Bits 23:16 | Command | Definition Chapter |
|------------------------------|----------------------------------|--|---------------------------|
| 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 | 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 | |
| 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 |

| Opcode Bits 26:24 | Sub Opcode Bits 23:16 | Command | Definition Chapter |
|----------------------|--------------------------|--|--------------------|
| 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 | SWTESS BASE ADDRESS | 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 | |

| Pipeline Type (28:27) | Opcode | Sub Opcode | Command | Definition Chapter |
|-----------------------|------------|------------|----------|--------------------|
| Common (pipelined) | Bits 26:24 | Bits 23:16 | | |
| 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 |
| | | | | |

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 |
| 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 |

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

Content for this topic is currently under development.

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 a 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:

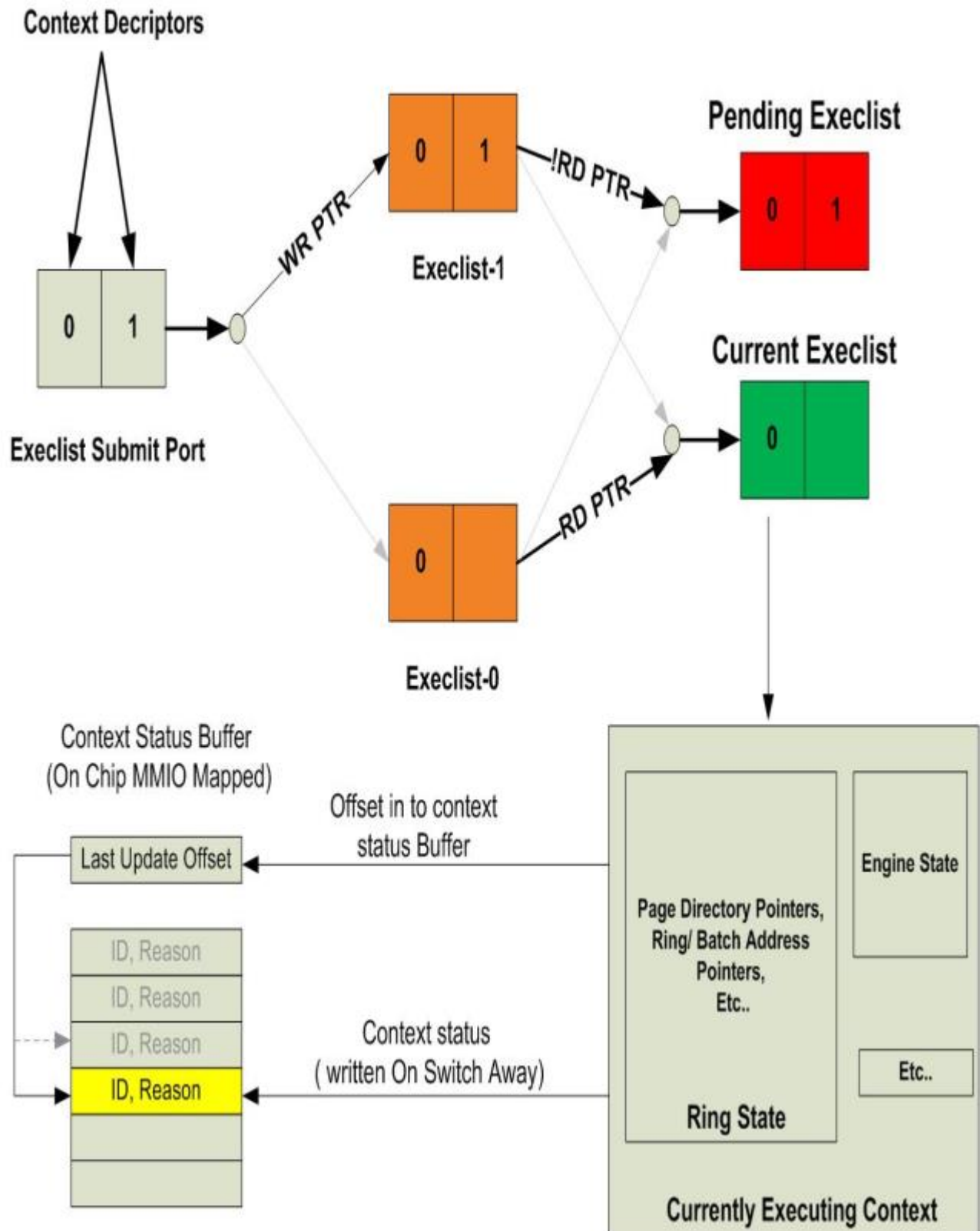
Execlist Submit Port Register

Execlist 0 Contents Register

Execlist 1 Contents

EXECLIST_STATUS - Execlist Status Register

Execlist Structure [CHV]



Logical Ring Context Format

When execlists are enabled, the Context Image for the each engine (Render, Video, Blitter, Video Enhancement) consists of Per-Process HW Status Page, Ring Context and Engine Context:

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

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

The detailed format of the Blitter/Video/VideoEnhancement logical ring context is documented in the "GPU Overview" volume, "Memory Data Formats" chapter.

The detailed format of the Render logical Ring and Engine Context including their size is mentioned in section "Graphics BSpec: 3D-Media-GPGPU Engine > [Render Command Memory Interface](#) > [Render Engine Logical Context Data](#) > Register/State Context" 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 |
| 28 | POSH Context Complete States the state of the POSH pipeline on a context switch by render pipe. When set indicates POSH pipe has completely processed (Head eqv to Tail) the submitted workload. |
| 27:24 | Reserved |
| 23:20 | Reserved |
| 19:16 | 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 when set to "0000b" indicates display plane on which Wait on Scanline/V-Blank/Sync Flip has been executed is indicated on bits [14:12]. 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 (future Sprite A) (Look at field 14:12) 0001b: Reserved (future Sprite B) 0010b: Reserved (future Sprite C) 0011b: Display Plane Sprite A2 0100b: Display Plane Sprite B2 0101b: Display Plane Sprite C2 0110b: Display Plane Sprite A3 0111b: Display Plane Sprite B3 1000b: Display Plane Sprite C3 1001b to 1111b: Reserved |

| Bits | Description |
|-------|--|
| 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. |
| 14:12 | <p>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.</p> <p>000b: Display Plane-A 001b: Display Plane-B 010b: Display Plane-C 011b: Display Plane Sprite-A 100b: Display Plane Sprite-B 101b: Display Plane Sprite-C</p> |
| 14:12 | <p>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.</p> <p>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.</p> <p>000b: Display Plane-A 001b: Display Plane-B 010b: Display Plane-C 011b: Display Plane Sprite-A 100b: Display Plane Sprite-B 101b: Display Plane Sprite-C</p> |
| 11 | <p>Semaphore Wait Mode</p> <p>0: Signal Mode 1: Poll Mode</p> <p>This field is valid and must be looked at only when the "Wait on Semaphore" field is set.</p> |
| 10 | Reserved |
| 9 | Reserved |
| 8 | <p>Wait on Scanline has resulted in context switch.</p> <p>This may be cleared during context exit if the condition is met. If none of the bits specifying why we exit and ring is not complete, then scheduler should reschedule the context when possible.</p> |
| 8 | Wait on Scanline has resulted in context switch. |

| Bits | Description |
|------|---|
| 7 | Wait on Semaphore has resulted in context switch. This may be cleared during context exit if the condition is met. If none of the bits specifying why we exit and ring is not complete, then scheduler should reschedule the context when possible. |
| 7 | Wait on Semaphore has resulted in context switch. |
| 6 | Wait on V-Blank has resulted in context switch. This may be cleared during context exit if the condition is met. If none of the bits specifying why we exit and ring is not complete, then scheduler should reschedule the context when possible. |
| 6 | Wait on V-Blank has resulted in context switch. |
| 5 | Wait on Sync Flip has resulted in context switch. This may be cleared during context exit if the condition is met. If none of the bits specifying why we exit and ring is not complete, then scheduler should reschedule the context when possible. |
| 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 |
| 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

| Project | Number of Status Entries |
|---------|--------------------------|
| CHV | 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_PTR - Context Status Buffer Pointer Register

CTXT_ST_BUF - Context Status Buffer Contents

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

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.

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 |
|--|
| <ul style="list-style-type: none"> • 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. |
| <ul style="list-style-type: none"> • 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:

INSTPM - Instruction Parser Mode Register

EXCC - Execute Condition Code Register

NOPIID - NOP Identification Register

CSPREEMPT - CSPREEMPT

RCS_CTXID_PREEMPTION_HINT - RCS Context Preemption Hint

RCS_PREEMPTION_HINT - RCS_PREEMPTION_HINT

RCS_PREEMPTION_HINT_UDW - RCS_PREEMPTION_HINT_UDW

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

CS_RESET_CTRL - CS Reset Control Register

HWS_PGA - Hardware Status Page Address Register

Hardware Status Page Layout

Logical Context Support

The following are the Logical Context Support Registers:

BB_ADDR - Batch Buffer Head Pointer Register

BB_ADDR_UDW - Batch Buffer Upper Head Pointer Register

RCS_BB_STATE - RCS Batch Buffer State Register

CTXT_SR_CTL - Context Save/Restore Control Register

CCID - Current Context Register

CXT_SIZE - Context Sizes

RS_CXT_OFFSET - Resource Streamer Context Offset

URB_CXT_OFFSET - URB Context Offset

CXT_EL_OFFSET - Exec-List 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

WAIT_FOR_RC6_EXIT - Control Register for Power Management

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

DISPLAY_MESSAGE_FORWARD_STATUS - Display Message Forward Status Register

R_PWR_CLK_STATE - Render Power Clock State Register

Context Save Registers

The following are the Context Save Registers:

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 Head Pointer Register for Upper DWord

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

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.

MI_NOOP

MI_ARB_CHECK

MI_ARB_ON_OFF

MI_BATCH_BUFFER_START

MI_CONDITIONAL_BATCH_BUFFER_END

MI_DISPLAY_FLIP

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_IMM

MI_LOAD_REGISTER_REG

MI_LOAD_REGISTER_MEM

MI_STORE_REGISTER_MEM

MI_SUSPEND_FLUSH

MI_UPDATE_GTT

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 buffer. 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 will execute the Privilege command as expected.

User Mode Privileged Commands

| User Mode Privileged Command | Function in Non-Privileged Batch Buffers | | | | |
|------------------------------|---|------------------|--|-----------------|-------------------------------|
| MI_UPDATE_GTT | Command is converted to NOOP. | | | | |
| MI_STORE_DATA_IMM | Command is converted to NOOP if Use Global GTT is enabled. | | | | |
| MI_STORE_DATA_INDEX | Command is converted to NOOP. | | | | |
| MI_STORE_REGISTER_MEM | Register read is always performed. Memory update is dropped if Use Global GTT is enabled. | | | | |
| MI_BATCH_BUFFER_START | <p>Command when executed from a batch buffer can set its "Privileged" level to its parent batch buffer or lower.</p> <p>Chained or Second level batch buffer can be "Privileged" only if the parent or the initial batch buffer is "Privileged". This is HW enforced.</p> | | | | |
| MI_LOAD_REGISTER_IMM | Command is converted to NOOP if the register accessed is privileged. | | | | |
| MI_LOAD_REGISTER_MEM | <p>Command is converted to NOOP if Use Global GTT is enabled.</p> <p>Command is converted to NOOP, if the register accessed is privileged.</p> | | | | |
| MI_LOAD_REGISTER_REG | Register write to a Privileged Register will be discarded. | | | | |
| MI_REPORT_PERF_COUNT | <p>Command is converted to NOOP if Use Global GTT is enabled</p> <p>If a privilege access violation occurs, the Report ID field in the report generated by the next legitimate MI_REPORT_PERF_COUNT may be corrupted.</p> | | | | |
| PIPE_CONTROL | <p>Still send flush down, Post-Sync Operation is NOOP if Use Global GTT or Use "Store Data Index" is enabled.</p> <p>Post-Sync Operation LRI to Privileged Register is discarded.</p> <table border="1" data-bbox="626 1690 1479 1785"> <tr> <th colspan="2">Programming Note</th></tr> <tr> <td>Context:</td><td>User Mode Privileged Commands</td></tr> </table> | Programming Note | | Context: | User Mode Privileged Commands |
| Programming Note | | | | | |
| Context: | User Mode Privileged Commands | | | | |

| User Mode Privileged Command | Function in Non-Privileged Batch Buffers |
|---------------------------------|---|
| | Command Privilege Violation error will not be set in a predictable fashion on a Privilege Access Violation from a PIPE_CONTROL command. |
| MI_SET_CONTEXT | Command is converted to NOOP. |
| MI_ATOMIC | Command is converted to NOOP if Use Global GTT is enabled. |
| MI_COPY_MEM_MEM | Command is converted to NOOP if Use Global GTT is used for source or destination address. |
| MI_SEMAPHORE_WAIT | Command is converted to NOOP if Use Global GTT is enabled. |
| MI_ARB_ON_OFF | Command is converted to NOOP. |
| MI_DISPLAY_FLIP | Command is converted to NOOP. |
| MI_CONDITIONAL_BATCH_BUFFER_END | Command is converted to NOOP if Use Global GTT is enabled. |

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

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

User Mode Non-Privileged Registers

| MMIO Name | MMIO Offset | SIZE in DWords |
|-----------------------|-------------|----------------|
| NOPID | 0x2094 | 1 |
| MI_PREDICATE_RESULT_2 | 0x23BC | 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 |
| CL_INVOCATION_COUNT | 0x2338 | 2 |
| CL_PRIMITIVES_COUNT | 0x2340 | 2 |
| PS_INVOCATION_COUNT_0 | 0x22C8 | 2 |
| PS_DEPTH_COUNT_0 | 0x22D8 | 2 |
| GPUGPU_DISPATCHDIMX | 0x2500 | 1 |
| GPUGPU_DISPATCHDIMY | 0x2504 | 1 |
| GPUGPU_DISPATCHDIMZ | 0x2508 | 1 |
| MI_PREDICATE_SRC0 | 0x2400 | 1 |
| MI_PREDICATE_SRC0 | 0x2404 | 1 |

| MMIO Name | MMIO Offset | SIZE in DWords |
|--------------------------|-------------|----------------|
| MI_PREDICATE_SRC1 | 0x2408 | 1 |
| MI_PREDICATE_SRC1 | 0x240C | 1 |
| MI_PREDICATE_DATA | 0x2410 | 1 |
| MI_PREDICATE_DATA | 0x2414 | 1 |
| MI_PRED_RESULT | 0x2418 | 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 |
| GPGPU_THREADS_DISPATCHED | 0x2290 | 2 |
| PS_INVOCATION_COUNT_1 | 0x22F0 | 2 |
| PS_DEPTH_COUNT_1 | 0x22F8 | 2 |
| BB_OFFSET | 0x2158 | 1 |
| MI_PREDICATE_RESULT_1 | 0x241C | 1 |
| CS_GPR (1-16) | 0x2600 | 32 |
| PS_INVOCATION_COUNT_2 | 0x2448 | 2 |
| PS_DEPTH_COUNT_2 | 0x2450 | 2 |
| 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 |
| SO_NUM_PRIMS_WRITTEN0 | 0x5200 | 2 |
| 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 |

| MMIO Name | MMIO Offset | SIZE in DWords |
|------------------|-------------|----------------|
| SO_WRITE_OFFSET3 | 0x528C | 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 |

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.

PR_CTR - Render Watchdog Counter

PR_CTR_THRSH - Render Watchdog Counter Threshold

Command Stream Front End

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 - Render

Bit Definition for Interrupt Control Registers - Blitter

Bit Definition for Interrupt Control Registers Media#1 (VDBOX1)

Bit Definition for Interrupt Control Registers Media#2 (VDBOX2)

Bit Definition for Interrupt Control Registers- Video Enhancement(VDBOX)

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 |

RCS_HWSTAM - Render Hardware Status Mask Register

RCS_IMR - Render Interrupt Mask Register

VCS_HWSTAM - VCS Hardware Status Mask Register

VCS_IMR - VCS Interrupt Mask Register

BCS_HWSTAM - BCS Hardware Status Mask Register

BCS_IMR - BCS Interrupt Mask Register

VECS_HWSTAM - VECS Hardware Status Mask Register

VECS_IMR - VECS 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 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:

RCS_EIR - Error Identity Register

RCS_EMR - Error Mask Register

RCS_ESR - Error Status Register

BCS_EIR - BCS Error Identity Register

BCS_EMR - BCS Error Mask Register

BCS_ESR - BCS Error Status Register

VCS_EIR - VCS Error Identity Register

VCS_EMR - VCS Error Mask Register

VCS_ESR - VCS Error Status Register

VECS_EIR - VECS Error Identity Register

VECS_EMR - VECS Error Mask Register

VECS_ESR - VECS 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 |
|---|
| <ul style="list-style-type: none"> • 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. |
| <ul style="list-style-type: none"> • 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 othe 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.

Watchdog Timer Registers

These are the Watchdog Timer registers:

BCS_CNTR - BCS Counter for the Blitter Engine

BCS_CTR_THRSH - BCS Watchdog Counter Threshold

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

This section defines the Hardware-Detected Error bit definitions and ordering that are 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 causes the Master Error bit in the ISR to be set. EIR bits remain set until the appropriate bits in the EIR are cleared by writing the appropriate EIR bits with 1 (except for the unrecoverable bits described below).

The following structure describes the Hardware-Detected Error bits:

BCS Hardware-Detected Error Bit Definitions Structure

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

BCS_EIR - BCS Error Identity Register

BCS_EMR - BCS Error Mask Register

BCS_ESR - BCS Error Status Register

Logical Context Support

The following are the Logical Context Support Registers:

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

CTXT_SR_CTL - Context Save/Restore Control 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

DISPLAY_MESSAGE_FORWARD_STATUS - Display Message Forward Status Register

BB_ADDR_DIFF - Batch Address Difference 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:

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

| Programming Note | |
|--|---|
| Context: | Mode Registers in Blitter Engine Command Streamer |
| <ul style="list-style-type: none"> 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. All Reserved bits are implemented. | |

BCS_EXCC - BCS Execute Condition Code Register

BCS_IDLELY - BCS Idle Switch Delay

BCS_SEMA_WAIT_POLL - BCS Semaphore Polling Interval on Wait

BCS_RESET_CTRL - BCS Reset Control Register

BCS_PREEMPTION_HINT - BCS_PREEMPTION_HINT

BCS_PREEMPTION_HINT_UDW - BCS_PREEMPTION_HINT_UDW

BCS_CTXID_PREEMPTION_HINT - BCS Context ID Preemption Hint

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.

MI_NOOP

MI_ARB_CHECK

MI_ARB_ON_OFF

MI_BATCH_BUFFER_START

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 BLT_MODE register. When privilege access checks are disabled HW executes privileged commands as expected.

User Mode Privileged Commands

| User Mode Privileged Command | Function in Non-Privileged Batch Buffers |
|------------------------------|--|
| MI_UPDATE_GTT | Command is converted to NOOP. |
| MI_STORE_DATA_IMM | Command is converted to NOOP if Use Global GTT is enabled. |
| MI_STORE_DATA_INDEX | Command is converted to NOOP. |
| MI_STORE_REGISTER_MEM | Command is converted to NOOP if Use Global GTT is enabled. |
| 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. |
| MI_LOAD_REGISTER_IMM | Command is converted to NOOP if the register accessed is privileged. |
| MI_LOAD_REGISTER_MEM | Command is converted to NOOP if Use Global GTT is enabled. Register write to a Privileged Register is discarded. |
| MI_LOAD_REGISTER_REG | Register write to a Privileged Register is discarded. |

| | |
|---------------------------------|--|
| MI_ATOMIC | Command is converted to NOOP if Use Global GTT is enabled. |
| MI_COPY_MEM_MEM | Command is converted to NOOP if Use Global GTT is used for source or destination address. |
| MI_SEMAPHORE_WAIT | Command is converted to NOOP if Use Global GTT is enabled. |
| MI_SEMAPHORE_SIGNAL | Command is converted to NOOP. |
| MI_ARB_ON_OFF | Command is converted to NOOP. |
| MI_DISPLAY_FLIP | Command is converted to NOOP. |
| MI_FLUSH_DW | Command is converted to NOOP if Use Global GTT or Use "Store Data Index" is enabled. |
| MI_CONDITIONAL_BATCH_BUFFER_END | Command is converted to NOOP if Use Global GTT is enabled. |

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 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 |

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_IMM

MI_STORE_DATA_INDEX

MI_ATOMIC

MI_COPY_MEM_MEM

MI_LOAD_REGISTER_IMM

MI_LOAD_REGISTER_REG

MI_LOAD_REGISTER_MEM

MI_STORE_REGISTER_MEM

MI_SUSPEND_FLUSH

MI_UPDATE_GTT

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 buffer. Batch buffers in GGTT memory space are privileged and batch buffers in PPGTT memory space are non-privileged. On parsing a 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 BLT_MODE register. When privilege access checks are disabled HW executes privileged commands as expected.

User Mode Privileged Commands

| User Mode Privileged Command | Function in Non-Privileged Batch Buffers |
|---------------------------------|--|
| MI_UPDATE_GTT | Command is converted to NOOP. |
| MI_STORE_DATA_IMM | Command is converted to NOOP if Use Global GTT is enabled. |
| MI_STORE_DATA_INDEX | Command is converted to NOOP. |
| MI_STORE_REGISTER_MEM | Command is converted to NOOP if Use Global GTT is enabled. |
| 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. |
| MI_LOAD_REGISTER_IMM | Command is converted to NOOP if the register accessed is privileged. |
| MI_LOAD_REGISTER_MEM | Command is converted to NOOP if Use Global GTT is enabled. Register write to a Privileged Register is discarded. |
| MI_LOAD_REGISTER_REG | Register write to a Privileged Register is discarded. |
| MI_ATOMIC | Command is converted to NOOP if Use Global GTT is enabled. |
| MI_COPY_MEM_MEM | Command is converted to NOOP if Use Global GTT is used for source or destination address. |
| MI_SEMAPHORE_WAIT | Command is converted to NOOP if Use Global GTT is enabled. |
| MI_SEMAPHORE_SIGNAL | Command is converted to NOOP. |
| MI_ARB_ON_OFF | Command is converted to NOOP. |
| MI_DISPLAY_FLIP | Command is converted to NOOP. |
| 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. |
| MI_CONDITIONAL_BATCH_BUFFER_END | Command is converted to NOOP if Use Global GTT is enabled. |

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

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

User Mode Non-Privileged Registers

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

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:

VCS_CNTR - VCS Counter for the bit stream decode engine

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

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

This section defines the Hardware-Detected Error bit definitions and ordering that are 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 causes the Master Error bit in the ISR to be set. EIR bits remain set until the appropriate bits in the EIR are cleared by writing the appropriate EIR bits with 1 (except for the unrecoverable bits described below).

The following links describe the 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

CTXT_SR_CTL - Context Save/Restore Control 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

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

Mode Registers

The following are Mode Registers:

VCS_CXT_SIZE - VCS Context Sizes]

VCS_MI_MODE - VCS Mode Register for Software Interface

MFV_MODE - Video Mode Register

VCS_INSTPM - VCS Instruction Parser Mode Register

VCS_IDLELY - VCS Idle Switch Delay

VCS_RESET_CTRL - VCS Reset Control Register

VCS_CTXID_PREEMPTION_HINT - VCS Context ID Preemption Hint

VCS_PREEMPTION_HINT - VCS_PREEMPTION_HINT

VCS_PREEMPTION_HINT_UDW

VCS_SEMA_WAIT_POLL - VCS Semaphore Polling Interval on Wait

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.

VCS_TIMESTAMP - VCS Reported Timestamp Count

VCS_CTX_TIMESTAMP - VCS 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_ARB_CHECK

MI_BATCH_BUFFER_END

MI_CONDITIONAL_BATCH_BUFFER_END

MI_BATCH_BUFFER_START

MI_FLUSH_DW

MI_LOAD_REGISTER_IMM

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_UPDATE_GTT

MI_LOAD_REGISTER_MEM[CHV]

MI_ATOMIC

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 MFX_MODE register. When privilege access checks are disabled HW will execute the Privilege command as expected.

User Mode Privileged Commands

| User Mode Privileged Command | Function in Non-Privileged Batch Buffers |
|---------------------------------|--|
| MI_UPDATE_GTT | Command is converted to NOOP. |
| MI_STORE_DATA_IMM | Command is converted to NOOP if Use Global GTT is enabled. |
| MI_STORE_DATA_INDEX | Command is converted to NOOP. |
| MI_STORE_REGISTER_MEM | Command is converted to NOOP if Use Global GTT is enabled. |
| 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. |
| MI_LOAD_REGISTER_IMM | Command is converted to NOOP if the register accessed is privileged. |
| MI_LOAD_REGISTER_MEM | Command is converted to NOOP if Use Global GTT is enabled Register write to a Privileged Register will be discarded. |
| MI_LOAD_REGISTER_REG | Register write to a Privileged Register will be discarded. |
| MI_ATOMIC | Command is converted to NOOP if Use Global GTT is enabled. |
| MI_COPY_MEM_MEM | Command is converted to NOOP if Use Global GTT is used for source or destination address. |
| MI_SEMAPHORE_SIGNAL | Command is converted to NOOP. |
| MI_SEMAPHORE_WAIT | Command is converted to NOOP if Use Global GTT is enabled. |
| MI_ARB_ON_OFF | Command is converted to NOOP. |
| 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. |
| MI_CONDITIONAL_BATCH_BUFFER_END | Command is converted to NOOP if Use Global GTT is enabled. |
| MI_SEMAPHORE_SIGNAL | Command is converted to NOOP. |

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 following table lists the non-privileged registers that can be written to from a non-secure batch buffer

executed from Video Command Streamer.

| User Mode Non-Privileged Registers | | |
|------------------------------------|-------------|----------------|
| MMIO Name | MMIO Offset | Size in DWords |
| VCS_GPR | 12600h | 32 |
| VCS_GPR (2nd VCS) | 1C600h | 32 |
| MFC_VDBOX1 | 12800h | 64 |
| MFC_VDBOX2 | 1C800h | 64 |

| User Mode Non-Privileged Registers | | |
|------------------------------------|---------------|----------------|
| MMIO Name | MMIO Range | Size in DWords |
| VCS_GPR | 12600h | 32 |
| VCS_GPR (2nd VCS) | 1C600h | 32 |
| MFC_VDBOX1 | 12800h-12FFFh | 512 |
| HuC | 0D0A0h | 4 |
| HuC | 0D064h | 1 |
| HuC | 0D080h | 1 |
| | | |
| HEVC | 1E900h | 64 |

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

Watchdog Timer Registers

The following are Watchdog Timer Registers:

VECS_CNTR - VECS Counter for the Video Enhancement Engine

VECS_CTR_THRSH - VECS Threshold for the Counter of Video Enhancement Engine

Logical Context Support

The following are Logical Context Support Registers:

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

CTXT_SR_CTL - Context Save/Restore Control Register

VECS_TIMESTAMP - VECS Reported Timestamp CountCHV

VECS_CTX_TIMESTAMP - VECS Context Timestamp Count

BB_ADDR_DIFF - Batch Address Difference 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

Mode Registers

The following are Mode Registers:

VECS_MI_MODE — VECS Mode Register for Software Interface

VEBOX_MODE - Video Mode Register

VECS_INSTPM—VECS Instruction Parser Mode Register

HWS_PGA - Hardware Status Page Address Register

Hardware Status Page Layout

VECS_RESET_CTRL - VECS Reset Control Register

VECS_PREEMPTION_HINT

VECS_PREEMPTION_HINT_UDW

VECS_CTXID_PREEMPTION_HINT - VECS Context ID Preemption Hint

VECS_SEMA_WAIT_POLL - VECS Semaphore Polling Interval on Wait

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_ARB_CHECK

MI_BATCH_BUFFER_END

MI_CONDITIONAL_BATCH_BUFFER_END

MI_BATCH_BUFFER_START

MI_FLUSH_DW

MI_LOAD_REGISTER_IMM

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_UPDATE_GTT

MI_COPY_MEM_MEM

MI_LOAD_REGISTER_MEM

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 a privileged command from a non-privileged batch buffer, a Command Privilege Violation Error is flagged and the command is dropped. A Command Privilege Violation Error is logged in the Error identity register of the 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 VEBOX_MODE register. When privilege access checks are disabled HW executes privileged commands as expected.

User Mode Privileged Commands

| User Mode Privileged Command | Function in Non-Privileged Batch Buffers |
|---------------------------------|--|
| MI_UPDATE_GTT | Command is converted to NOOP. |
| MI_STORE_DATA_IMM | Command is converted to NOOP if Use Global GTT is enabled. |
| MI_STORE_DATA_INDEX | Command is converted to NOOP. |
| MI_STORE_REGISTER_MEM | Command is converted to NOOP if Use Global GTT is enabled. |
| 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. |
| MI_LOAD_REGISTER_IMM | Command is converted to NOOP if the register accessed is privileged. |
| MI_LOAD_REGISTER_MEM | Command is converted to NOOP if Use Global GTT is enabled. Register write to a Privileged Register will be discarded. |
| MI_LOAD_REGISTER_REG | Register write to a Privileged Register will be discarded. |
| MI_ATOMIC | Command is converted to NOOP if Use Global GTT is enabled. |
| MI_COPY_MEM_MEM | Command is converted to NOOP if Use Global GTT is used for source or destination address. |
| MI_SEMAPHORE_SIGNAL | Command is converted to NOOP. |
| MI_SEMAPHORE_WAIT | Command is converted to NOOP if Use Global GTT is enabled. |
| MI_ARB_ON_OFF | Command is converted to NOOP. |
| 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. |
| MI_CONDITIONAL_BATCH_BUFFER_END | Command is converted to NOOP if Use Global GTT is enabled. |

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

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

User Mode Non-Privileged Registers

| MMIO Name | MMIO Offset | Size in DWords |
|-----------|-------------|----------------|
| VECS_GPR | 1A600h | 32 |

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_CH ECK | Elemen t Bound ary | Semapho re Wait | Wait for Event | 3DPRIMIT IVE | GPGPU_WA LKER | PIPE_CONT ROL *** | MEDI A STATE FLUS H |
| Render | AP | AP | Unsucces sful & AP | Unsucces sful & AP | Object Level (if enabled *) | Mid-Thread (if enabled **) | PIPESEL- GPGPU MODE | PIPESE L- GPGP U MODE |
| Blitter | AP | AP | Unsucces sful & AP | Unsucces sful & AP | N/A | N/A | N/A | N/A |
| Media | AP | AP | Unsucces sful & AP | N/A | N/A | N/A | N/A | N/A |
| Video Enhancem ent | AP | AP | Unsucces sful & 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.

**** Any Header with the value [31:29] = "011", [28:27] = "00" OR "11" and [26:24] = "001". Refer to [Graphics Command Formats](#)

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