

# Intel Open Source Graphics Programmer's Reference Manual (PRM) for the 2013 Intel® Core™ Processor Family, including Intel HD Graphics, Intel Iris™ Graphics and Intel Iris Pro Graphics

## Volume 6: Command Stream Programming (Haswell)

1/21/2014

# Copyright

# Command Stream Programming

## Table of Contents

# Graphics Command Formats

This section describes the general format of the graphics device commands.

Graphics commands are defined with various formats. The first DWord of all commands is called the *header* DWord. The header contains the only field common to all commands, the *client* field that determines the device unit that processes the command data. The Command Parser examines the client field of each command to condition the further processing of the command and route the command data accordingly.

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<br>00h – NOP<br>0Xh – Single DWord Commands<br>1Xh – Two+ DWord Commands<br>2Xh – Store Data Commands<br>3Xh – Ring/Batch Buffer Cmds | | Identification No./DWord Count<br>Command Dependent Data<br>5:0 – DWord Count<br>5:0 – DWord Count<br>5:0 – DWord Count | | |
| Reserved | 001, 010 | Opcode – 11111 | 23:19<br>Sub Opcode 00h – 01h | | 18:16<br>Re-served | 15:0<br>DWord Count |
| | | | | | | |
| **TYPE** | **31:29** | **28:27** | **26:24** | **23:16** | **15:8** | **7:0** |
| Common | 011 | 00 | Opcode – 000 | Sub Opcode | Data | DWord Count |
| Common (NP) | 011 | 00 | Opcode – 001 | Sub Opcode | Data | DWord Count |
| Reserved | 011 | 00 | Opcode – 010 – 111 | | | |
| Single Dword Command | 011 | 01 | Opcode – 000 – 001 | Sub Opcode | | N/A |
| Reserved | 011 | 01 | Opcode – 010 – 111 | | | |
| Media State | 011 | 10 | Opcode – 000 | Sub Opcode | | Dword Count |
| Media Object | 011 | 10 | Opcode – 001 – 010 | Sub Opcode | Dword Count | |
| Reserved | 011 | 10 | Opcode – 011 – 111 | | | |
| 3DState | 011 | 11 | Opcode – 000 | Sub Opcode | Data | DWord Count |
| 3DState (NP) | 011 | 11 | Opcode – 001 | Sub Opcode | Data | DWord Count |
| PIPE_Control | 011 | 11 | Opcode – 010 | | Data | DWord Count |
| 3DPrimitive | 011 | 11 | Opcode – 011 | | Data | DWord Count |
| Reserved | 011 | 11 | Opcode – 100 – 111 | | | |
| Reserved | 100 | XX | | | | |
| Reserved | 101 | XX | | | | |

| Bits | | | | | | | |
|---|---|---|---|---|---|---|---|
| TYPE | 31:29 | 28:24 | | 23 | 22 | 21:0 | |
| Reserved | 110 | XX | | | | | |
| Fulsim[2] | 111 | XX | | | | | |

Notes:

1. The qualifier "NP" indicates that the state variable is non-pipelined and the render pipe is flushed before such a state variable is updated. The other state variables are pipelined (default).
2. [31:29] == '111' is reserved for fulsim command decodings. It is invalid for HW to parse this command.

### Video Command Header Format

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

| Bits | | | | | | | |
|---|---|---|---|---|---|---|---|
| TYPE | 31:29 | 28:24 | | 23 | 22 | 21:0 | |
| 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 | | | |

## Memory Interface Commands

Memory Interface (MI) commands are basically those commands which do not require processing by the 2D or 3D Rendering/Mapping engines. The functions performed by these commands include:

- Control of the command stream (e.g., Batch Buffer commands, breakpoints, ARB On/Off, etc.)
- Hardware synchronization (e.g., flush, wait-for-event)
- Software synchronization (e.g., Store DWORD, report head)
- Graphics buffer definition (e.g., Display buffer, Overlay buffer)
- Miscellaneous functions

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

### Memory Interface Commands for RCP

| Opcode (28:23) | Command | Pipe | | | |
|---|---|---|---|---|---|
| | | Render | Video | Blitter | Video Enhancements |
| 1-DWord | | | | | |
| 00h | MI_NOOP | All | All | All | All |
| 01h | MI_SET_PREDICATE | | | | |
| 02h | MI_USER_INTERRUPT | All | All | All | All |
| 03h | MI_WAIT_FOR_EVENT | All | All | All | All |
| 05h | MI_ARB_CHECK | All | All | All | All |
| 06h | MI_RS_CONTROL | | | | |
| 07h | MI_REPORT_HEAD | All | All | All | All |
| 08h | MI_ARB_ON_OFF | | | | |
| 09h | MI_URB_ATOMIC_ALLOC | | | | |
| 0Ah | MI_BATCH_BUFFER_END | All | All | All | All |
| 0Bh | MI_SUSPEND_FLUSH | All | | | |
| 0Ch | MI_PREDICATE | | | | |
| 0Dh | MI_TOPOLOGY_FILTER | | | | |
| 0Eh | MI_SET_APPID | | [DevIVB+] | | |

| Opcode (28:23) | Command | Pipe | | | |
|---|---|---|---|---|---|
| | | Render | Video | Blitter | Video Enhancements |
| **1-DWord** | | | | | |
| 0Fh | MI_RS_CONTEXT | [DevHSW+] | | | |
| **2+ DWord** | | | | | |
| 10h | Reserved | | | | |
| 14h | MI_DISPLAY_FLIP [HSW] | All | | | |
| 15h | Reserved | | | | |
| 17h | Reserved | | | | |
| 18h | MI_SET_CONTEXT | All | | | |
| 1Ah | MI_MATH | | | | |
| 1Eh–1Fh | Reserved | | | | |
| **Store Data** | | | | | |
| 20h | MI_STORE_DATA_IMM | All | All | All | All |
| 21h | MI_STORE_DATA_INDEX | All | All | All | All |
| 22h | MI_LOAD_REGISTER_IMM | All | All | All | All |
| 24h | MI_STORE_REGISTER_MEM | All | All | All | All |
| 27h | MI_CLFLUSH | | | | |
| 28h | MI_REPORT_PERF_COUNT | | | | |
| 2Bh | MI_RS_STORE_DATA_IMM | | | | |
| 2Ch | MI_LOAD_URB_MEM | | | | |
| 2Dh | MI_STORE_URM_MEM | | | | |
| **Ring/Batch Buffer** | | | | | |
| 30h | Reserved | | | | |
| 31h | MI_BATCH_BUFFER_START | All | All | All | All |
| 32h–35h | Reserved | | | | |
| 37h–3Fh | Reserved | | | | |

## 2D Commands

The 2D commands include various flavors of BLT operations, along with commands to set up BLT engine state without actually performing a BLT. Most commands are of fixed length, though there are a few commands that include a variable amount of "inline" data at the end of the command.

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

### 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 |

| Opcode (28:22) | Command |
|---|---|
| 5Ah-70h | Reserved |
| 71h | XY_MONO_SRC_COPY_IMMEDIATE_BLT |
| 72h | XY_PAT_BLT_IMMEDIATE |
| 73h | XY_SRC_COPY_CHROMA_BLT |
| 74h | XY_FULL_IMMEDIATE_PATTERN_BLT |
| 75h | XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT |
| 76h | XY_PAT_CHROMA_BLT |
| 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 [HSW] | 3D Pipeline |
| 0h | 05h | 3DSTATE_DEPTH_BUFFER [HSW] | 3D Pipeline |
| 0h | 06h | 3DSTATE_STENCIL_BUFFER [HSW] | 3D Pipeline |
| 0h | 07h | 3DSTATE_HIER_DEPTH_BUFFER [HSW] | 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 [HSW] | 3D Pipeline |
| 0h | 0Eh | 3DSTATE_CC_STATE_POINTERS [HSW] | 3D Pipeline |
| 0h | 10h | 3DSTATE_VS [HSW] | Vertex Shader |
| 0h | 11h | 3DSTATE_GS [HSW] | Geometry Shader |
| 0h | 12h | 3DSTATE_CLIP [HSW] | Clipper |
| 0h | 13h | 3DSTATE_SF [HSW] | Strips & Fans |
| 0h | 14h | 3DSTATE_WM [HSW] | Windower |
| 0h | 15h | 3DSTATE_CONSTANT_VS [HSW] | Vertex Shader |
| 0h | 16h | 3DSTATE_CONSTANT_GS [HSW] | Geometry Shader |
| 0h | 17h | 3DSTATE_CONSTANT_PS [HSW] | Windower |
| 0h | 18h | 3DSTATE_SAMPLE_MASK [HSW] | Windower |
| 0h | 19h | 3DSTATE_CONSTANT_HS [HSW] | Hull Shader |
| 0h | 1Ah | 3DSTATE_CONSTANT_DS [HSW] | Domain Shader |
| 0h | 1Bh | 3DSTATE_HS [HSW] | Hull Shader |
| 0h | 1Ch | 3DSTATE_TE [HSW] | Tesselator |
| 0h | 1Dh | 3DSTATE_DS [HSW] | Domain Shader |

| Opcode Bits 26:24 | Sub Opcode Bits 23:16 | Command | Definition Chapter |
|---|---|---|---|
| 0h | 1Eh | 3DSTATE_STREAMOUT [HSW] | HW Streamout |
| 0h | 1Fh | 3DSTATE_SBE [HSW] | Setup |
| 0h | 20h | 3DSTATE_PS [HSW] | Pixel Shader |
| 0h | 21h | 3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP [HSW] | Strips & Fans |
| 0h | 22h | Reserved | |
| 0h | 23h | 3DSTATE_VIEWPORT_STATE_POINTERS_CC [HSW] | Windower |
| 0h | 24h | 3DSTATE_BLEND_STATE_POINTERS [HSW] | Pixel Shader |
| 0h | 25h | 3DSTATE_DEPTH_STENCIL_STATE_POINTERS [HSW] | Pixel Shader |
| 0h | 26h | 3DSTATE_BINDING_TABLE_POINTERS_VS [HSW] | Vertex Shader |
| 0h | 27h | 3DSTATE_BINDING_TABLE_POINTERS_HS [HSW] | Hull Shader |
| 0h | 28h | 3DSTATE_BINDING_TABLE_POINTERS_DS [HSW] | Domain Shader |
| 0h | 29h | 3DSTATE_BINDING_TABLE_POINTERS_GS [HSW] | Geometry Shader |
| 0h | 2Ah | 3DSTATE_BINDING_TABLE_POINTERS_PS [HSW] | Pixel Shader |
| 0h | 2Bh | 3DSTATE_SAMPLER_STATE_POINTERS_VS [HSW] | Vertex Shader |
| 0h | 2Ch | 3DSTATE_SAMPLER_STATE_POINTERS_HS [HSW] | Hull Shader |
| 0h | 2Dh | 3DSTATE_SAMPLER_STATE_POINTERS_DS [HSW] | Domain Shader |
| 0h | 2Eh | 3DSTATE_SAMPLER_STATE_POINTERS_GS [HSW] | Geometry Shader |
| 0h | 2Fh | Reserved | |
| 0h | 30h | 3DSTATE_URB_VS [HSW] | Vertex Shader |
| 0h | 31h | 3DSTATE_URB_HS [HSW] | Hull Shader |
| 0h | 32h | 3DSTATE_URB_DS [HSW] | Domain Shader |
| 0h | 33h | 3DSTATE_URB_GS [HSW] | Geometry Shader |
| 0h | 34h | 3DSTATE_GATHER_CONSTANT_VS [HSW] | Vertex Shader |
| 0h | 35h | 3DSTATE_GATHER_CONSTANT_GS [HSW] | Geometry Shader |
| 0h | 36h | 3DSTATE_GATHER_CONSTANT_HS [HSW] | Hull Shader |
| 0h | 37h | 3DSTATE_GATHER_CONSTANT_DS [HSW] | Domain Shader |
| 0h | 38h | 3DSTATE_GATHER_CONSTANT_PS [HSW] | Pixel Shader |
| 0h | 39h | 3DSTATE_DX9_CONSTANTF_VS [HSW] | Vertex Shader |
| 0h | 3Ah | 3DSTATE_DX9_CONSTANTF_PS [HSW] | Pixel Shader |
| 0h | 3Bh | 3DSTATE_DX9_CONSTANTI_VS [HSW] | Vertex Shader |
| 0h | 3Ch | 3DSTATE_DX9_CONSTANTI_PS [HSW] | Pixel Shader |
| 0h | 3Dh | 3DSTATE_DX9_CONSTANTB_VS [HSW] | Vertex Shader |
| 0h | 3Eh | 3DSTATE_DX9_CONSTANTB_PS [HSW] | Pixel Shader |
| 0h | 3Fh | 3DSTATE_DX9_LOCAL_VALID_VS [HSW] | Vertex Shader |
| 0h | 40h | 3DSTATE_DX9_LOCAL_VALID_PS [HSW] | Pixel Shader |

| Opcode Bits 26:24 | Sub Opcode Bits 23:16 | Command | Definition Chapter |
|---|---|---|---|
| 0h | 41h | 3DSTATE_DX9_GENERATE_ACTIVE_VS [HSW] | Vertex Shader |
| 0h | 42h | 3DSTATE_DX9_GENERATE_ACTIVE_PS [HSW] | Pixel Shader |
| 0h | 43h | 3DSTATE_BINDING_TABLE_EDIT_VS [HSW] | Vertex Shader |
| 0h | 44h | 3DSTATE_BINDING_TABLE_EDIT_GS [HSW] | Geometry Shader |
| 0h | 45h | 3DSTATE_BINDING_TABLE_EDIT_HS [HSW] | Hull Shader |
| 0h | 46h | 3DSTATE_BINDING_TABLE_EDIT_DS [HSW] | Domain Shader |
| 0h | 47h | 3DSTATE_BINDING_TABLE_EDIT_PS [HSW] | Pixel Shader |
| 0h | 48h-4Bh | Reserved [HSW] | |
| 0h | 4Ch | 3DSTATE_WM_CHROMA_KEY | Windower |
| 0h | 4Dh | 3DSTATE_PS_BLEND | Windower |
| 0h | 4Eh | 3DSTATE_WM_DEPTH_STENCIL | Windower |
| 0h | 4Fh | 3DSTATE_PS_EXTRA | Windower |
| 0h | 50h | 3DSTATE_RASTER | Strips & Fans |
| 0h | 51h | 3DSTATE_SBE_SWIZ | Strips & Fans |
| 0h | 52h | 3DSTATE_WM_HZ_OP | Windower |
| 0h | 53h | 3DSTATE_INT (internally generated state) | 3D Pipeline |
| 0h | 56h-FFh | Reserved | |
| 1h | 00h | 3DSTATE_DRAWING_RECTANGLE | Strips & Fans |
| 1h | 02h | 3DSTATE_SAMPLER_PALETTE_LOAD0 | Sampling Engine |
| 1h | 03h | Reserved | |
| 1h | 04h | 3DSTATE_CHROMA_KEY | Sampling Engine |
| 1h | 05h | Reserved [HSW] | |
| 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 [HSW] | Windower |
| 1h | 0Bh | 3DSTATE_GS_SVB_INDEX [HSW] | Geometry Shader |
| 1h | 0Ch | 3DSTATE_SAMPLER_PALETTE_LOAD1 [HSW] | Sampling Engine |
| 1h | 0Dh | 3DSTATE_MULTISAMPLE [HSW] | Windower |
| 1h | 0Eh | 3DSTATE_STENCIL_BUFFER [HSW] | Windower |
| 1h | 0Fh | 3DSTATE_HIER_DEPTH_BUFFER [HSW] | Windower |
| 1h | 10h | 3DSTATE_CLEAR_PARAMS [HSW] | Windower |
| 1h | 11h | 3DSTATE_MONOFILTER_SIZE [HSW] | Sampling Engine |
| 1h | 12h | 3DSTATE_PUSH_CONSTANT_ALLOC_VS [HSW] | Vertex Shader |
| 1h | 13h | 3DSTATE_PUSH_CONSTANT_ALLOC_HS [HSW] | Hull Shader |

| Opcode Bits 26:24 | Sub Opcode Bits 23:16 | Command | Definition Chapter |
|---|---|---|---|
| 1h | 14h | 3DSTATE_PUSH_CONSTANT_ALLOC_DS [HSW] | Domain Shader |
| 1h | 15h | 3DSTATE_PUSH_CONSTANT_ALLOC_GS [HSW] | Geometry Shader |
| 1h | 16h | 3DSTATE_PUSH_CONSTANT_ALLOC_PS [HSW] | Pixel Shader |
| 1h | 17h | 3DSTATE_SO_DECL_LIST | HW Streamout |
| 1h | 18h | 3DSTATE_SO_BUFFER | HW Streamout |
| 1h | 19h | 3DSTATE_BINDING_TABLE_POOL_ALLOC [HSW] | Resource Streamer |
| 1h | 1Ah | 3DSTATE_GATHER_POOL_ALLOC [HSW] | Resource Streamer |
| 1h | 1Bh | 3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC [HSW] | 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 | 04h–FFh | Reserved | n/a |
| **Reserved** | **Bits 26:24** | **Bits 23:16** | | |
| 0h | 2h–7h | XX | Reserved | n/a |

## MFX Commands

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

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

| Pipeline Type (28:27) | Opcode (26:24) | 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 |

| Pipeline Type (28:27) | Opcode (26:24) | SubopA (23:21) | SubopB (20:16) | Command | Chapter | Recommended Indirect State Pointer Map | Interruptable? |
|---|---|---|---|---|---|---|---|
| 2h | 1h | 1h | 8h | MFD_AVC_BSD_OBJECT | MFX | N/A | No |
| 2h | 1h | 1h | 9-1Fh | Reserved | N/A | N/A | N/A |
| AVC Enc | | | | | | | |
| 2h | 1h | 2h | 0-1h | Reserved | N/A | N/A | N/A |
| 2h | 1h | 2h | 2h | MFC_AVC_FQM_STATE | MFX | IMAGE | N/A |
| 2h | 1h | 2h | 3-7h | Reserved | N/A | N/A | N/A |
| 2h | 1h | 2h | 8h | MFC_AVC_PAK_INSERT_OBJECT | MFX | N/A | N/A |
| 2h | 1h | 2h | 9h | MFC_AVC_PAK_OBJECT | MFX | N/A | Yes |
| 2h | 1h | 2h | A-1Fh | Reserved | N/A | N/A | N/A |
| 2h | 1h | 2h | 0-1Fh | Reserved | N/A | N/A | N/A |
| VC1 Common | | | | | | | |
| 2h | 2h | 0h | 0h | MFX_VC1_PIC_STATE | MFX | IMAGE | N/A |
| 2h | 2h | 0h | 1h | MFX_VC1_PRED_PIPE_STATE | MFX | IMAGE | N/A |
| 2h | 2h | 0h | 2h | MFX_VC1_DIRECTMODE_STATE | MFX | SLICE | N/A |
| 2h | 2h | 0h | 2-1Fh | Reserved | N/A | N/A | N/A |
| VC1 Dec | | | | | | | |
| 2h | 2h | 1h | 0-7h | Reserved | N/A | N/A | N/A |
| 2h | 2h | 1h | 8h | MFD_VC1_BSD_OBJECT | MFX | N/A | Yes |
| 2h | 2h | 1h | 9-1Fh | Reserved | N/A | N/A | N/A |
| VC1 Enc | | | | | | | |
| 2h | 2h | 2h | 0-1Fh | Reserved | N/A | N/A | N/A |
| MPEG2Common | | | | | | | |
| 2h | 3h | 0h | 0h | MFX_MPEG2_PIC_STATE | MFX | IMAGE | N/A |
| 2h | 3h | 0h | 1h | MFX_MPEG2_QM_STATE | MFX | IMAGE | N/A |
| 2h | 3h | 0h | 2-1Fh | Reserved | N/A | N/A | N/A |
| MPEG2 Dec | | | | | | | |
| 2h | 3h | 1h | 1-7h | Reserved | N/A | N/A | N/A |
| 2h | 3h | 1h | 8h | MFD_MPEG2_BSD_OBJECT | MFX | N/A | Yes |
| 2h | 3h | 1h | 9-1Fh | Reserved | N/A | N/A | N/A |
| MPEG2 Enc | | | | | | | |
| 2h | 3h | 2h | 0-1Fh | Reserved | N/A | N/A | N/A |
| The Rest | | | | | | | |
| 2h | 4-5h, | x | x | Reserved | N/A | N/A | N/A |

| Pipeline Type (28:27) | Opcode (26:24) | SubopA (23:21) | SubopB (20:16) | Command | Chapter | Recommended Indirect State Pointer Map | Interruptable? |
|---|---|---|---|---|---|---|---|
| | 7h | | | | | | |

# Blitter Engine Command Interface

## BCS_RINGBUF—Ring Buffer Registers

Following is a list of ring buffer registers:

**RING_BUFFER_TAIL - Ring Buffer Tail**

**RING_BUFFER_HEAD - Ring Buffer Head**

**RING_BUFFER_START - Ring Buffer Start**

**RING_BUFFER_CTL - Ring Buffer Control**

**UHPTR - Pending Head Pointer Register**

# Blitter Engine Command Interface

## BCS_RINGBUF—Ring Buffer Registers

Following is a list of ring buffer registers:

**RING_BUFFER_TAIL - Ring Buffer Tail**

**RING_BUFFER_HEAD - Ring Buffer Head**

**RING_BUFFER_START - Ring Buffer Start**

**RING_BUFFER_CTL - Ring Buffer Control**

**UHPTR - Pending Head Pointer Register**

## BLT Watchdog Timer Registers

These are the Watchdog Timer registers:

**BCS_CTR_THRSH - BCS Watchdog Counter Threshold**

**PR_CTR_THRSH - Watchdog Counter Threshold**

**PR_CTR_CTL - Watchdog Counter Control**

## BLT Interrupt Control Registers

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

### Bit Definition for Interrupt Control Registers

| Bit | Description |
|---|---|
| 31:30 | **Reserved. MBZ:** These bits may be assigned to interrupts on future products/steppings. |
| 29 | **Page Fault:**<br>[Pre-DevHSW,DevHSW:A]: This bit is set whenever there is a pending page or directory fault in blitter command streamer.<br>[DevHSW,EXCLUDE(DevHSW:A)]: This bit is set whenever there is a pending GGTT/PPGTT (page or directory) fault in Blitter command streamer when Fault Repair Mode is disabled.<br>On Fault Repair mode Enabled, this bit will never get set and will get collapsed with the Render command streamer page fault error.<br>Please refer to vol1c "page fault support" section for more details. |
| 28:27 | **Reserved. MBZ** |
| 26 | **MI_FLUSH_DW Notify Interrupt:** The Pipe Control packet (Fences) specified in *3D pipeline* document may optionally generate an Interrupt. The Store QW associated with a fence is completed ahead of the interrupt. |
| 25 | **Blitter Command Parser Master Error:** When this status bit is set, it indicates that the hardware has detected an error. It is set by the device upon an error condition and cleared by a CPU write of a one to the appropriate bit contained in the Error ID register followed by a write of a one to this bit in the IIR. Further information on the source of the error comes from the "Error Status Register" which along with the "Error Mask Register" determine which error conditions will cause the error status bit to be set and the interrupt to occur.<br>**Page Table Error:** Indicates a page table error.<br>**Instruction Parser Error:** The Blitter Instruction Parser encounters an error while parsing an instruction. |
| 24 | **Sync Status:** This bit is set when the Instruction Parser completes a flush with the sync enable bit active in the INSTPM register. The event will happen after all the blitter engines are flushed. The HW Status DWord write resulting from this event will cause the CPU's view of graphics memory to be coherent as well (flush and invalidate the blitter cache). It is the driver's responsibility to clear this bit before the next sync flush with HWSP write enabled. |
| 23 | **Reserved. MBZ** |
| 22 | **Blitter Command Parser User Interrupt:** This status bit is set when an MI_USER_INTERRUPT instruction is executed on the Render Command Parser. Note that instruction execution is not halted and proceeds normally. A mechanism such as an MI_STORE_DATA instruction is required to associate a particular meaning to a user interrupt. |
| 21:0 | **Reserved. MBZ** |

**BCS_HWSTAM - BCS Hardware Status Mask Register**

**BCS_IMR - BCS Interrupt Mask Register**

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

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

The following structure describes the Hardware-Detected Error bits:

**BCS Hardware-Detected Error Bit Definitions**

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

**BCS_EIR - BCS Error Identity Register**

**BCS_EMR - BCS Error Mask Register**

**BCS_ESR - BCS Error Status Register**

## BLT Logical Context Support

Following are the Logical Context Support Registers:

**BB_ADDR - Batch Buffer Head Pointer Register**

**SBB_ADDR - Second Level Batch Buffer Head Pointer 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**

**SBB_PREEMPT_ADDR - Second Level Batch Buffer Head Pointer Preemption Register**

**MI_PREDICATE_RESULT_1 - Predicate Rendering Data Result 1**

## Mode Registers

The following are Mode Registers:

**BCS_MI_MODE - BCS Mode Register for Software Interface**

**BLT_MODE - Blitter Mode Register**

**BCS_INSTPM - BCS Instruction Parser Mode Register**

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

**Programming Notes:**

- All Reserved bits are implemented.

**BCS_EXCC - BCS Execute Condition Code Register**

**BRSYNC - Blitter/Render Semaphore Sync Register**

**BVSYNC - Blitter/Video Semaphore Sync Register**

**BVESYNC - Blitter/Video Enhancement Semaphore Sync Register**

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

**HWS_PGA - Hardware Status Page Address Register**

**Hardware Status Page Layout**

## MI Commands for Blitter Engine

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

**MI_NOOP**

**MI_ARB_CHECK**

**MI_ARB_ON_OFF**

**MI_BATCH_BUFFER_START**

**MI_BATCH_BUFFER_END**

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

**MI_NOOP**

**MI_ARB_CHECK**

**MI_ARB_ON_OFF**

**MI_BATCH_BUFFER_START**

## Command Access to Privileged Memory

Memory space mapped through the global GTT is considered "privileged" memory. Commands that have the capability of accessing both privileged and unprivileged (PPGTT space) memory will contain a bit that, if set, will attempt a "privileged" access through the GGTT rather than an unprivileged access through the context-local PPGTT.

"User mode" command buffers should not be able to access privileged memory under any circumstances. These command buffers will be issued by the kernel mode driver with the batch buffer's **Buffer Security** Indicator set to "non-secure". Commands in such a batch buffer are not allowed to access privileged memory. The commands in these buffers are supplied by the user mode driver and will not be validated by the kernel mode driver. For a batch buffer marked as non-secure if **Per-Process Virtual Address Space** is set, the command buffer fetches are generated using the PPGTT space.

"Kernel mode" command buffers are allowed to access privileged memory. The batch buffers Buffer Security indicator is set to "secure" in this case. In some of the commands that access memory in a secure batch buffer, a bit is provided in the command to steer the access to Per process or Global virtual space. Secure batch buffers are executed from the global GTT.

Commands in ring buffers and commands in batch buffers that are marked as secure (by the kernel mode driver) are allowed to access both privileged and unprivileged memory and may choose on a command-by-command basis.

### GGTT and PPGTT Usage by Command

| Command | Address | Allowed Access |
|---|---|---|
| MI_BATCH_BUFFER_START* | Command Address | Selectable |
| MI_DISPLAY_FLIP | Display Buffer Base | GGTT Only |
| MI_STORE_DATA_IMM* | Storage Address | Selectable |
| MI_STORE_DATA_INDEX** | Storage Offset | Selectable |
| MI_STORE_REGISTER_MEM* | Storage Address | Selectable |
| MI_SEMAPHORE_MBOX | Semaphore Address | Selectable |
| PIPE_CONTROL | STDW Address | Selectable |

*Command has a GGTT/PPGTT selector added to it vs. previous Gen family products.

**Added bit allows offset to apply to global HW Status Page or PP HW Status Page found in context image.

## Privileged Commands

A subset of the commands are privileged. These commands may be issued only from a secure batch buffer or directly from a ring. If one of these commands is parsed in a non-secure batch buffer, an error is flagged and the command is dropped. For commands that generates a write, hardware completes the transaction but the byte enables are turned off. Batch buffers from the User mode driver are passed directly to the kernel mode driver which does not validate them but issues them with the Security Indicator set to 'non-secure' to protect the system from an attack using these privileged commands.

### Privileged Commands

| Privileged Command | Function in Non-Privileged Batch Buffers |
|---|---|
| MI_LOAD_REGISTER_IMM | Byte enables are turned off. |
| MI_UPDATE_GTT | Byte enabled are turned off. |
| MI_STORE_DATA_IMM | Command is translated using the Per-process GTT if **Per-Process Virtual Address Space** is set. |
| MI_STORE_DATA_INDEX | Command is translated using the Per process hardware status page if **Per-Process Virtual Address Space Enable** is set. |
| MI_STORE_REGISTER_MEM | Command is translated and completed with byte enables turned off. |
| MI_DISPLAY_FLIP | Command is ignored by the hardware. |

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

## User Mode Privileged Commands

A subset of the commands are privileged. These commands may be issued only from a secure batch buffer or directly from a ring. If one of these commands is parsed in a non-secure batch buffer, an error is flagged and the command is dropped. For commands that generates a write, the hardware will complete the transaction but the byte enables are turned off. Batch buffers from the User mode driver are passed directly to the kernel mode driver which does not validate them but issues them with the Security Indicator set to 'non-secure' to protect the system from an attack using these privileged commands.

### User Mode Privileged Commands

| User Mode Privileged Command | Function in non-privileged batch buffers |
|---|---|
| MI_LOAD_REGISTER_IMM | Command is converted to NOOP |
| 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 if **Use Global GTT** is enabled. |
| MI_STORE_REGISTER_MEM | Command is converted to NOOP |
| MI_DISPLAY_FLIP | Command is converted to NOOP |
| MI_ARB_ON_OFF | Command is converted to NOOP |
| MI_ARB_CHECK | Command is converted to NOOP |
| MI_WAIT_FOR_EVENT | Command is converted to NOOP |

## User Mode Privileged Commands

A subset of the commands are privileged. These commands may be issued only from a secure batch buffer or directly from a ring. If one of these commands is parsed in a non-secure 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.

Batch buffers from the User mode driver are passed directly to the kernel mode driver which does not validate them but issues them with the Security Indicator set to 'non-secure' to protect the system from an attack using these privileged commands.

### 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 if **Use Global GTT** is enabled. |
| MI_STORE_REGISTER_MEM | Register read is always performed. Memory update is dropped if **Use Global GTT** is enabled. |
| MI_LOAD_REGISTER_MEM | Command is converted to NOOP. |
| MI_BATCH_BUFFER_START | Command when executed from a batch buffer can set its "Privileged" level to its parent batch buffer or lower.<br>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. |
| MI_REPORT_PERF_COUNT | Command is converted to NOOP if **Use Global GTT** is enabled. |
| PIPE_CONTROL | Still send flush down, Post-Sync Operation is NOOP if Use Global GTT is enabled.<br>LRI **Post-Sync Operation** is NOOP. |
| MI_SET_CONTEXT | Command is converted to NOOP. |
| MI_LOAD_REGISTER_REG | Register read is always performed. Memory update is dropped 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.

**MI_BATCH_BUFFER_END**

**MI_CONDITIONAL_BATCH_BUFFER_END**

**MI_DISPLAY_FLIP**

**MI_LOAD_SCAN_LINES_EXCL**

**MI_LOAD_SCAN_LINES_INCL**

**MI_FLUSH**

**MI_CLFLUSH**

**MI_MATH**

**MI_REPORT_HEAD**

**MI_STORE_DATA_IMM**

**MI_STORE_DATA_INDEX**

**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_MBOX**

## RINGBUF — Ring Buffer Registers

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

**RING_BUFFER_TAIL - Ring Buffer Tail**

**RING_BUFFER_HEAD - Ring Buffer Head**

**RING_BUFFER_START - Ring Buffer Start**

**RING_BUFFER_CTL - Ring Buffer Control**

**UHPTR - Pending Head Pointer Register**

## Render Watchdog Timer Registers

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

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

**PR_CTR_CTL - Watchdog Counter Control**

**PR_CTR_THRSH - Watchdog Counter Threshold**

**PR_CTR - Render Watchdog Counter**

## Render Interrupt Control Registers

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

### Bit Definition for Interrupt Control Registers

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

| Bit | Interrupt Bit | ISR bit Reporting via Hardware Status Write (when unmasked via HWSTAM) |
|---|---|---|
| 9 | **Performance Monitoring Buffer Half-Full Interrupt** | Set when event occurs, cleared when event cleared |
| 8 | **Context Switch Interrupt:** Set when a context switch has just occurred. | Not supported to be unmasked |
| 7 | **Page Fault:** This bit is set whenever there is a pending PPGTT (page or directory) fault. | Set when event occurs, cleared when event cleared<br>[HSW]: 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 |
| 5 | **L3 Parity interrupt** | |
| 4 | PIPE_CONTROL packet - Notify Enable | 0 |
| 3 | Master Error | Set when error occurs, cleared when error cleared |
| 2 | Sync Status | Toggled every SyncFlush Event |
| 1 | | |
| 0 | User Interrupt | 0 |

**HWSTAM - Hardware Status Mask Register**

**IMR - Interrupt Mask Register**

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

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

The following table describes the Hardware-Detected Error bits:

**Hardware-Detected Error Bits**

**Hardware-Detected Error Bit Definitions**

Following are the the EIR, EMR and ESR registers:

**EIR - Error Identity Register**

**EMR - Error Mask Register**

**ESR - Error Status Register**

## Logical Context Support

Following are the Logical Context Support Registers:

**BB_ADDR - Batch Buffer Head Pointer Register**

**RCS_BB_STATE - RCS Batch Buffer State Register**

**SBB_ADDR - Second Level Batch Buffer Head Pointer Register**

**SBB_STATE - Second Level Batch Buffer State Register**

## Context Save Registers

Following are the Context Save Registers:

**BB_PREEMPT_ADDR - Batch Buffer 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**

## Mode Registers

The following are the Mode Registers:

**INSTPM - Instruction Parser Mode Register**

**EXCC - Execute Condition Code Register**

**NOPID - NOP Identification Register**

**RVSYNC - Render/Video Semaphore Sync Register**

**RVESYNC - Render/Video Enhancement Semaphore Sync Register**

**RBSYNC - Render/Blitter Semaphore Sync Register**

**CTX_SEMA_REG - Context Semaphore Sync Registers**

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

**HWS_PGA - Hardware Status Page Address Register**

**Hardware Status Page Layout**

# MI Commands for 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 products within the Gen4+ family. However, slight changes may be present in some commands (i.e., for features added or removed), or some commands may be removed entirely. Refer to the *Preface* chapter for product specific summary.

## Command Access to Privileged Memory

Memory space mapped through the global GTT is considered "privileged" memory. Commands that have the capability of accessing both privileged and unprivileged (PPGTT space) memory will contain a bit that, if set, will attempt a "privileged" access through the GGTT rather than an unprivileged access through the context-local PPGTT.

"User mode" command buffers should not be able to access privileged memory under any circumstances. These command buffers will be issued by the kernel mode driver with the batch buffer's **Buffer Security** Indicator set to "non-secure". Commands in such a batch buffer are not allowed to access privileged memory. The commands in these buffers are supplied by the user mode driver and will not be validated by the kernel mode driver. For a batch buffer marked as non-secure if **Per-Process Virtual Address Space** is set, the command buffer fetches are generated using the PPGTT space.

"Kernel mode" command buffers are allowed to access privileged memory. The batch buffers Buffer Security indicator is set to "secure" in this case. In some of the commands that access memory in a secure batch buffer, a bit is provided in the command to steer the access to Per process or Global virtual space. Secure batch buffers are executed from the global GTT.

Commands in ring buffers and commands in batch buffers that are marked as secure (by the kernel mode driver) are allowed to access both privileged and unprivileged memory and may choose on a command-by-command basis.

### GGTT and PPGTT Usage by Command

| Command | Address | Allowed Access |
|---|---|---|
| MI_BATCH_BUFFER_START* | Command Address | Selectable |
| MI_DISPLAY_FLIP | Display Buffer Base | GGTT Only |
| MI_STORE_DATA_IMM* | Storage Address | Selectable |
| MI_STORE_DATA_INDEX** | Storage Offset | Selectable |
| MI_STORE_REGISTER_MEM* | Storage Address | Selectable |
| MI_SEMAPHORE_MBOX | Semaphore Address | Selectable |
| PIPE_CONTROL | STDW Address | Selectable |

*Command has a GGTT/PPGTT selector added to it vs. previous Gen family products.

**Added bit allows offset to apply to global HW Status Page or PP HW Status Page found in context image.

## User Mode Privileged Commands

A subset of the commands are privileged. These commands may be issued only from a secure batch buffer or directly from a ring. If one of these commands is parsed in a non-secure 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.

Batch buffers from the User mode driver are passed directly to the kernel mode driver which does not validate them but issues them with the Security Indicator set to 'non-secure' to protect the system from an attack using these privileged commands.

### 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 if **Use Global GTT** is enabled. |
| MI_STORE_REGISTER_MEM | Register read is always performed. Memory update is dropped if **Use Global GTT** is enabled. |
| MI_LOAD_REGISTER_MEM | Command is converted to NOOP. |
| MI_BATCH_BUFFER_START | Command when executed from a batch buffer can set its "Privileged" level to its parent batch buffer or lower.<br>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. |
| MI_REPORT_PERF_COUNT | Command is converted to NOOP if **Use Global GTT** is enabled. |
| PIPE_CONTROL | Still send flush down, Post-Sync Operation is NOOP if Use Global GTT is enabled.<br>LRI **Post-Sync Operation** is NOOP. |
| MI_SET_CONTEXT | Command is converted to NOOP. |
| MI_LOAD_REGISTER_REG | Register read is always performed. Memory update is dropped 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.

**MI_BATCH_BUFFER_END**

**MI_CONDITIONAL_BATCH_BUFFER_END**

**MI_DISPLAY_FLIP**

**MI_LOAD_SCAN_LINES_EXCL**

**MI_LOAD_SCAN_LINES_INCL**

**MI_FLUSH**

**MI_CLFLUSH**

**MI_MATH**

**MI_REPORT_HEAD**

**MI_STORE_DATA_IMM**

**MI_STORE_DATA_INDEX**

**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_MBOX**

## Video Command Streamer (VCS)

The VCS (Video Command Streamer) unit is primarily served 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 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 and 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.

## Video Command Streamer (VCS)

The VCS (Video Command Streamer) unit is primarily served 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 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 and 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.

## VCS_RINGBUF—Ring Buffer Registers

**RING_BUFFER_TAIL - Ring Buffer Tail**

**RING_BUFFER_HEAD - Ring Buffer Head**

**RING_BUFFER_START - Ring Buffer Start**

**RING_BUFFER_CTL - Ring Buffer Control**

**UHPTR - Pending Head Pointer Register**

## Watchdog Timer Registers

The following registers are defined as Watchdog Timer registers:

**VCS_CNTR - VCS Counter for the bit stream decode engine**

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

**VCS_CNTR - VCS Counter for the bit stream decode engine**

## Interrupt Control Registers

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

### Bit Definition for Interrupt Control Registers

| Bit | Description |
|---|---|
| 31:21 | **Reserved. MBZ:** These bits may be assigned to interrupts on future products/steppings. |
| 20 | **Context Switch Interrupt.** Set when a context switch has just occurred.<br>**Per-Process Virtual Address Space** bit needs to be set for this interrupt to occur. |
| 19 | **Page Fault:** This bit is set whenever there is a pending page or directory fault.<br>[DevHSW-B0+]: This bit is set whenever there is a pending GGTT/PPGTT (page or directory) fault in Video command streamer when Fault Repair Mode is disabled.<br>On Fault Repair mode Enabled, this bit will never get set and will get collapsed with the Render command streamer page fault error.<br>Please refer to vol1c "page fault support" section for more details. |
| 18 | **Timeout Counter Expired:** Set when the VCS timeout counter has reached the timeout thresh-hold value. |
| 17 | **Reserved** |
| 16 | **MI_FLUSH_DW Notify Interrupt:** The Pipe Control packet (Fences) specified in *3D pipeline* document may optionally generate an Interrupt. The Store QW associated with a fence is completed ahead of the interrupt. |
| 15 | **Video Command Parser Master Error:** When this status bit is set, it indicates that the hardware has detected an error. It is set by the device upon an error condition and cleared by a CPU write of a one to the appropriate bit contained in the Error ID register followed by a write of a one to this bit in the IIR. Further information on the source of the error comes from the "Error Status Register" which along with the "Error Mask Register" determine which error conditions will cause the error status bit to be set and the interrupt to occur.<br>**Page Table Error:** Indicates a page table error.<br>**Instruction Parser Error:** The Video Instruction Parser encounters an error while parsing an instruction. |
| 14 | **Sync Status:** This bit is set when the Instruction Parser completes a flush with the sync enable bit active in the INSTPM register. The event will happen after all the MFX engines are flushed. The HW Status DWord write resulting from this event will cause the CPU's view of graphics memory to be coherent as well (flush and invalidate the MFX cache).It is the driver's responsibility to clear this bit |

| Bit | Description |
|---|---|
| | before the next sync flush with HWSP write enabled |
| 12 | **Video Command Parser User Interrupt:** This status bit is set when an MI_USER_INTERRUPT instruction is executed on the Video Command Parser. Note that instruction execution is not halted and proceeds normally. A mechanism such as an MI_STORE_DATA instruction is required to associate a particular meaning to a user interrupt. |
| 11:0 | **Reserved:** MBZ |

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

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

### VCS_HWSTAM - VCS Hardware Status Mask Register

### VCS_IMR - VCS Interrupt Mask Register

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

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

The following table describes the Hardware-Detected Error bits:

### Hardware-Detected Error Bits

#### VCS Hardware-Detected Error Bit Definitions

#### VCS_EIR - VCS Error Identity Register

#### VCS_EMR - VCS Error Mask Register

#### VCS_ESR - VCS Error Status Register

## Logical Context Support

This section contains the registers for Logical Context Support.

**BB_STATE - Batch Buffer State Register**

**BB_ADDR - Batch Buffer Head Pointer Register**

**SBB_ADDR - Second Level Batch Buffer Head Pointer Register**

## Mode Registers

Following are Mode Registers:

**BBA_LEVEL2 - 2nd Level Batch Buffer Address**

**VCS_MI_MODE - VCS Mode Register for Software Interface**

**MFX_MODE - Video Mode Register**

**VCS_INSTPM - VCS Instruction Parser Mode Register**

**VBSYNC - Video/Blitter Semaphore Sync Register**

**VRSYNC - Video/Render Semaphore Sync Register**

**VVESYNC - Video Codec/Video Enhancement Semaphore Sync Register**

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

**HWS_PGA - Hardware Status Page Address Register**

**Hardware Status Page Layout**

# Registers in Media Engine

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

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

**MI_ARB_CHECK**

**MI_ARB_ON_OFF**

**MI_BATCH_BUFFER_END**

**MI_CONDITIONAL_BATCH_BUFFER_END**

**MI_BATCH_BUFFER_START**

**MI_FLUSH_DW**

**MI_LOAD_REGISTER_IMM**

**MI_NOOP**

**MI_REPORT_HEAD**

**MI_SEMAPHORE_MBOX**

**MI_STORE_REGISTER_MEM**

**MI_STORE_DATA_IMM**

**MI_STORE_DATA_INDEX**

**MI_SUSPEND_FLUSH**

**MI_USER_INTERRUPT**

**MI_UPDATE_GTT**

**MI_WAIT_FOR_EVENT**

**MI_LOAD_REGISTER_MEM**

## VECS_RINGBUF — Ring Buffer Registers

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**

**UHPTR - Pending Head Pointer Register**

## VECS_RINGBUF — Ring Buffer Registers

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**

**UHPTR - Pending Head Pointer Register**

## Watchdog Timer Registers

Following are Watchdog Timer Registers:

**VECS_CTR_THRSH - VECS Threshold for the Counter of Video Enhancement Engine**

## Interrupt Control Registers

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

### Bit Definition for Interrupt Control Registers [HSW]

| Bit | Description |
|---|---|
| 31:14 | **Reserved. MBZ:** These bits may be assigned to interrupts on future products/steppings. |
| 13 | MI_FLUSH_DW notify |
| 12 | VECS error interrupt |
| 11 | MMIO sync flush status |
| 10 | VECS MI_USER_INTERRUPT |
| 9:0 | **Reserved:** MBZ |

**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 table describes the Hardware-Detected Error bits:

### Hardware-Detected Error Bits

**VECS Hardware-Detected Error Bit Definitions**

**VECS_EIR - VECS Error Identity Register**

**VECS_EMR - VECS Error Mask Register**

**VECS_ESR - VECS Error Status Register**

## Logical Context Support

Following are Logical Context Support Registers:

**BB_ADDR - Batch Buffer Head Pointer Register**

**BB_STATE - Batch Buffer State Register**

**VECS_TIMESTAMP - VECS Reported Timestamp Count**

**VCS_TIMESTAMP - VCS Reported Timestamp Count**

## Mode Registers

Following are Mode Registers:

VECS_CXT_SIZE - VECS Context Sizes

**VECS_MI_MODE — VECS Mode Register for Software Interface**

**VEBOX_MODE - Video Mode Register**

**VECS_INSTPM—VECS Instruction Parser Mode Register**

**VECS_NOPID — VECS NOP Identification Register**

**VEBSYNC - Video/Blitter Semaphore Sync Register**

**VERSYNC - Video/Render Semaphore Sync Register**

**VEVSYNC - Video Enhancement/Video Semaphore Sync Register**

**HWS_PGA - Hardware Status Page Address Register**

**Hardware Status Page Layout**

## MI Commands for Video Enhancement Engine

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

This chapter describes MI Commands for the Video Codec Engine.

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

**MI_ARB_CHECK**

**MI_ARB_ON_OFF**

**MI_BATCH_BUFFER_END**

**MI_CONDITIONAL_BATCH_BUFFER_END**

**MI_BATCH_BUFFER_START**

**MI_FLUSH_DW**

**MI_LOAD_REGISTER_IMM**

**MI_NOOP**

**MI_REPORT_HEAD**

**MI_SEMAPHORE_MBOX**

**MI_STORE_REGISTER_MEM**

**MI_STORE_DATA_IMM**

**MI_STORE_DATA_INDEX**

**MI_SUSPEND_FLUSH**

**MI_USER_INTERRUPT**

**MI_UPDATE_GTT**

**MI_WAIT_FOR_EVENT**

**MI_LOAD_REGISTER_MEM**

## Resource Streamer

This topic is currently under development.

## Introduction

The resource streamer is added to offload work from the driver without compromising on GPU optimizations. In order to reduce latency associated with these offloaded operation, H/W adds a Resource Streamer. The Resource Streamer is almost S/W invisible; S/W sees a single command stream, but it may be best for the S/W to be aware that the RS is present, as certain operations might be emphasized. The resource streamer will run ahead of the 3D Command Streamer and process only the certain commands. The Cmd steamer processes these same commands for purposes of buffer full synchronization and buffer consumption.

## Common Abbreviations

| | |
|---|---|
| CS | **Command Streamer.** Block in charge of streaming commands. The Resource Streamer (RS) is primarily an accelerator for the CS. |
| FF | **Fixed Function.** Any fixed function hardward. |
| RS | **The Resource Streamer.** Responsible for reducing command latencies for certain command operations. |
| URB | **Unified Return Buffer.** The mechanism for returning information from a command. |

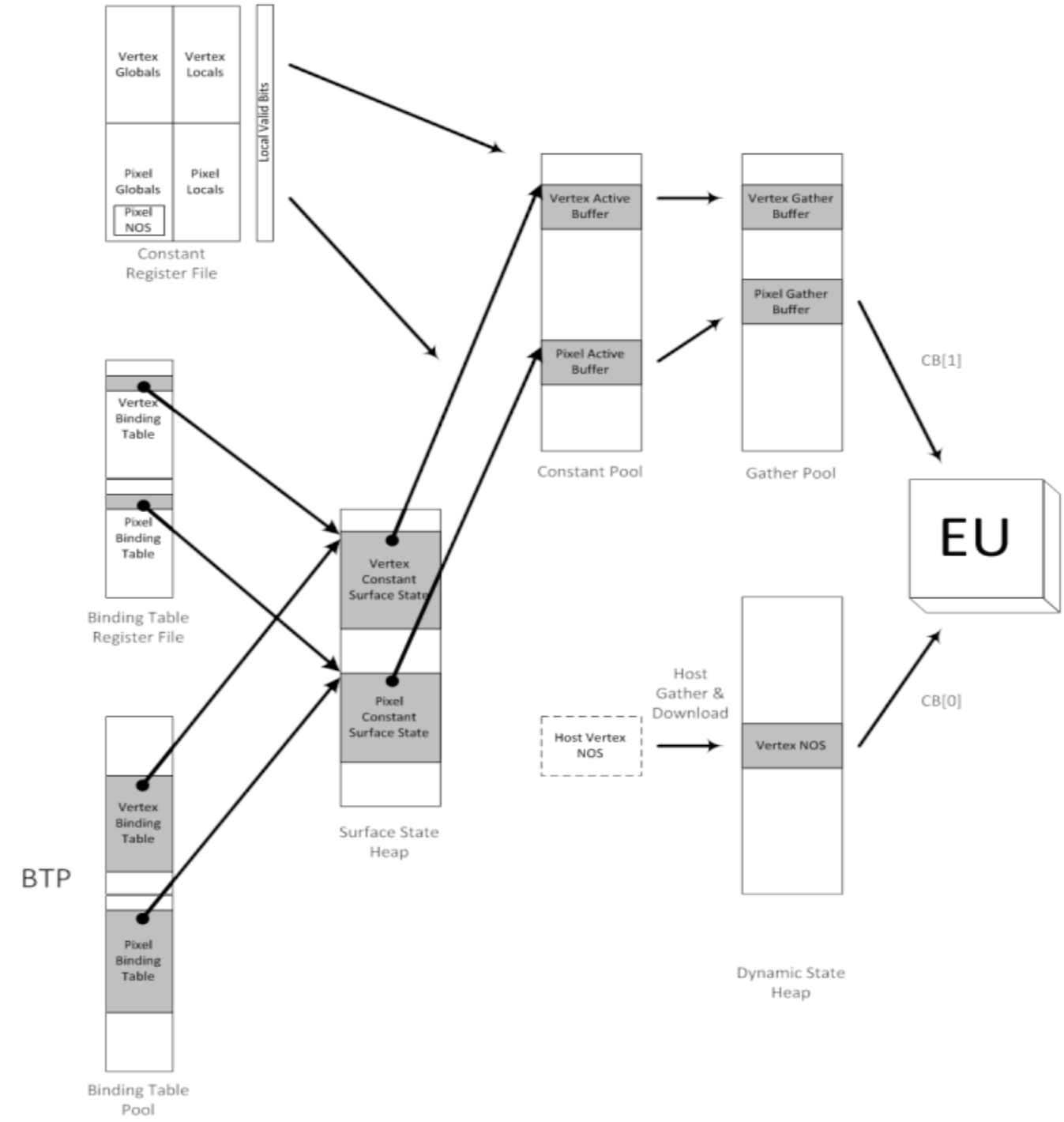
## Theory of Operation

This section briefly describes the operation of the Resource Streamer.  Specifically, it calls out reset state, initialization requirements, and major operational tasks of the RS.

## Resource Streamer Functions

The Resource Streamer (RS) examines the commands in the ring buffer in an attempt to pre-process certain long latency items for the remainder of the graphics processing. The RS is used for the following operations:

- Batch Processing – The resource streamer reads ahead of command streamer activity to unwind batch buffers.
- Context Save – When the Command Streamer signals that context must be saved, the RS makes certain all previous cycles are completed, saves all context, and signals completion to the command streamer.
- Gather Push Constants – The RS detects GATHER commands (**GATHER_POOL_ALLOC, GATHER_***), and prefetchs contents needed for further command processing. The RS gets the base address of the contents by detecting the **GATHER_POOL_ALLOC** command, and uses other **GATHER_*** commands to generate reads for data, and writes out data to memory.
- Constant Buffer Generation – Similar to other constant processes, the RS intercepts the commands for constants to update state and data.
- HW Binding Table Generation/Flush – The RS detects operations in the command stream to update binding table state and memory with bind table contents.




## Resource Streamer Activity Diagram

# Detailed Resource Streamer Operations

## Introduction

This chapter describes the operation of the Resource Streamer in deeper detail. Most of the operations of the Resource Streamer are processed from ring buffer shown in the Ring Buffer Organization Figure in Resource Streamer Operation Descriptions. The RS examines the command stream from the ring buffer to pre-process information required by the 3D Command Streamer (CS). For a large number of the commands, the RS takes no action.

## Resource Streamer Operation Descriptions

### Batch Processing

Batch processing is a method of extending the Ring Buffer by the insertion of additional commands. The Ring Buffer normally will process all commands in order stepping through each location in the buffer until the commands are complete. By inserting an MI_BATCH_BUFFER_START into the command stream, commands are fetched from a new location indicated in this command.

*Batch Processing* shows an example of a Ring Buffer that uses a batch buffer. The MI_BATCH_BUFFER_START command is obtained from Ring Buffer, and command processing continues in the new location. Batch buffers can be chained, as is shown in the diagram. At the end of the second batch buffer, the MI_BATCH_BUFFER_END command indicates that we are to return to processing from the Ring Buffer.

## Context Save

When the CS indicates that there is a context to be saved or restored, the RS saves its context. The CS provides and address for the RS image and issue a "batch buffer start" (see section *Batch Processing*). The RS will consume this image just like any other batch buffer, and stop when it reaches the MI_BATCH_BUFFER_END command.

The context image for the Resource streamer consists of the following components:

1. HW_BINDING_TABLE_IMAGE
2. GATHER_IMAGE
3. CONSTANT_IMAGE
4. MI_BATCH_BUFFER_END

These will be discussed in the follow subsections.

## HW Binding Table Image

While it is not always necessary to save binding table information, "split points" context switches must be saved, so the binding table contents are always saved. These consist of:

- Binding Table Generate Enable
- Binding Table Pool Base Address
- Binding Table Pool Size
- Binding Table Contents

### HW Binding Table Image

| Description | Dwords Required for Storage |
|---|---|
| 3DSTATE_BINDING_TABLE_POOL_ALLOC | 3 |
| 3DSTATE_BINDING_TABLE_EDIT_VS | 194 |
| 3DSTATE_BINDING_TABLE_EDIT_GS | 194 |
| 3DSTATE_BINDING_TABLE_EDIT_HS | 194 |
| 3DSTATE_BINDING_TABLE_EDIT_DS | 194 |
| 3DSTATE_BINDING_TABLE_EDIT_PS | 194 |
| 3DSTATE_BINDING_TABLE_EDIT_VS | 194 |

## Gather Push Constants Image

Since the resource streamer does not support mid-triangle preemption, the resource steamer will have finished producing all the gather buffers by the end of the batch buffer and the cmd streamer would have consumed all the gather buffers. The following things need to be saved.

- Gather pool enable
- Gather pool base address
- Gather pool size

Therefore a 3DSTATE_GATHER_POOL_ALLOC command needs to be saved.

### Gather Push Constants Image

| Description | Dwords Required for Storage |
|---|---|
| 3DSTATE_GATHER_POOL_ALLOC | 4 |

## Push Constant Image

We assume that the end of the batch buffer can come between any set of cmds. Therefore the following things will be saved:

- Dx9 Constant enable
- Dx9 Constant pool base address
- Dx9 Constant pool size
- Dx9 local registers (F,I,B)

- Dx9 Local Valid
- Dx9 global registers (F,I,B)

Therefore a 3DSTATE_CONSTANT_BUFFER_POOL_ALLOC command will saved. In addition, since the F register is 256 entries and only a maximum of 63 entries can be contained in a single 3DSTATE_DX9CONSTANTF_* command, 5 CONSTANTF cmds will be saved for global and 5 for local registers register per FF (VS,PS). There will be 1 3DSTATE_CONSTANTI_* will be save for global and 1 for local register per FF. There will be 1 3DSTATE_CONSTANTB_* will be save for global and 1 for local register per FF.

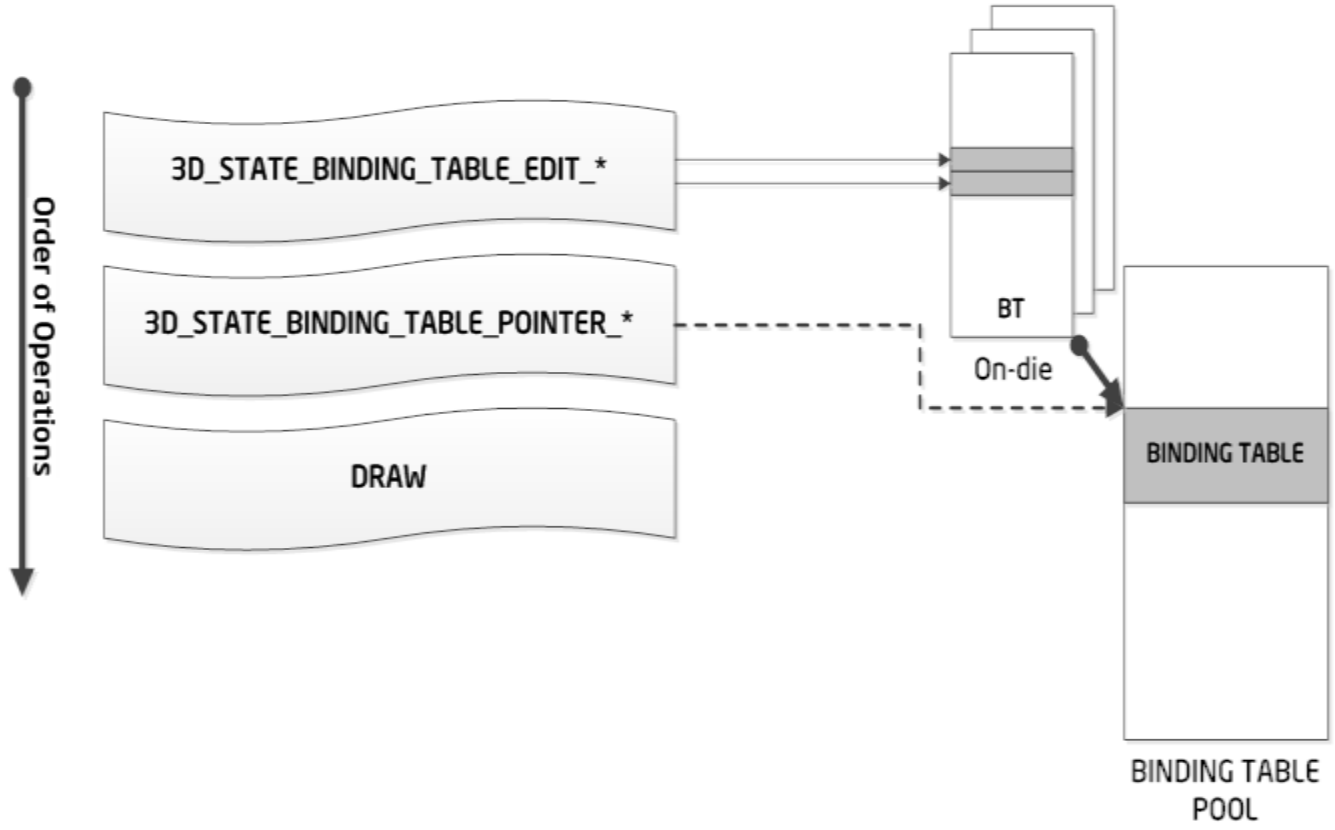
### Gather Push Constants Image

| Description | Dwords Required for Storage |
|---|---|
| 3DSTATE_CONSTANT_BUFFER_POOL_ALLOC | 4 |
| 3DSTATE_CONSTANTF_VS | 1026 |
| 3DSTATE_CONSTANTI_VS | 130 |
| 3DSTATE_CONSTANTB_VS | 18 |
| 3DSTATE_CONSTANTF_VS | 1026 |
| 3DSTATE_CONSTANTI_VS | 130 |
| 3DSTATE_CONSTANTB_VS | 18 |
| 3DSTATE_LOCAL_VALID_VS | 10 |
| 3DSTATE_CONSTANTF_PS | 1026 |
| 3DSTATE_CONSTANTI_PS | 130 |
| 3DSTATE_CONSTANTB_PS | 18 |
| 3DSTATE_CONSTANTF_PS | 1026 |
| 3DSTATE_CONSTANTI_PS | 130 |
| 3DSTATE_CONSTANTB_PS | 18 |
| 3DSTATE_LOCAL_VALID_PS | 10 |

## HW Binding Table Generation

The RS generates binding tables in hardware to offload this from the driver. There is an on-die set of binding tables for each fixed-function unit (VS, GS, HS, DS, PS). There are a set of commands generated by the driver for to update each of these tables (3D_STATE_BINDING_TABLE_POINTER_*). When the RS encounters any of these commands, it will write the corresponding binding table out to the binding table pool. When the CS encounters these commands, it will send the binding table points down as pipelined state.

### HW Binding Table Generation



| Project: | HSW |
|---|---|

The following table describes the different types of usages with binding table generation.

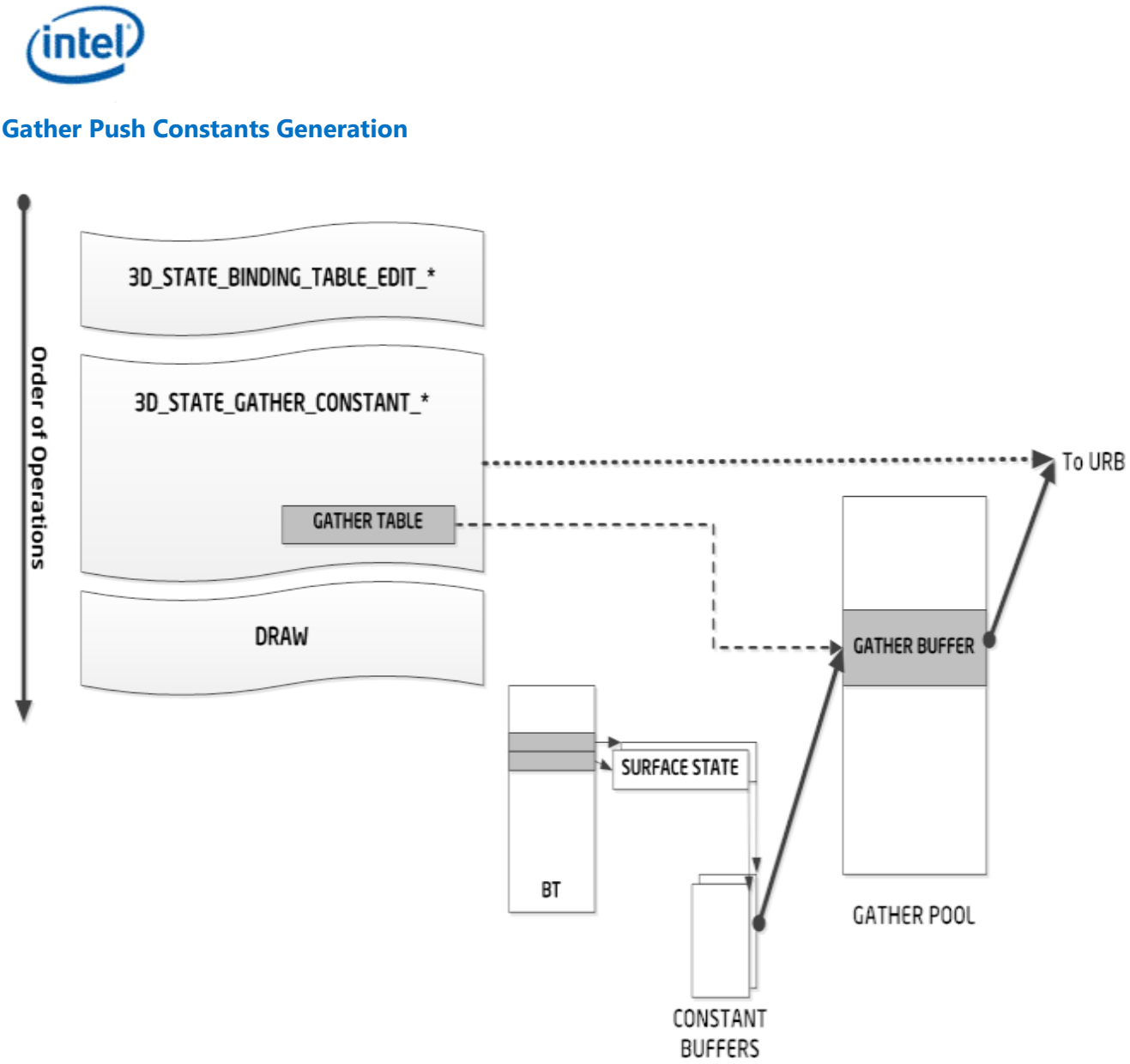
| RS active* | BT Pool Enabled | Mode |
|---|---|---|
| 0 | 0 | SW Generate BT in Surface State Heap |
| 0 | 1 | Illegal(Undefined) |
| 1 | 0 | Illegal** |
| 1 | 1 | HW Generate BT |
| * Active means with RS enabled in Batch buffer and MI_RS_CONTROL field with RS on. | | |
| ** If RS is enabled, Binding table pool is required to be enabled | | |

## Gather Push Constants

Applications can provide up to 16 constant buffers. The compiler does some optimizations of constant usage and determines which constants should be packed in which constant register for optimal shader performance. While this gathering and packing of constant elements into push constants optimizes the shaders, it causes the driver additional work at draw call time, since the driver must gather and pack the constants at draw time.

The RS offloads the gathering process for the driver by interpreting the 3D_STATE_GATHER_CONSTANT_* for each of the fixed functions (VS, GS, DS, HS, PS). The compiler generates at gather table which instructs which elements of the buffers should be packed into the gather buffer. The gather table indexes the binding table to get a surface state which in turn points to the constant buffer. Once the gather buffer has been filled, the CS will execute the 3D_STATE_GATHER_CONSTANT_* to load the push constant into the URB.

**NOTE: The gather push constants can ONLY BE USED if the HW generated binding tables are also used.**



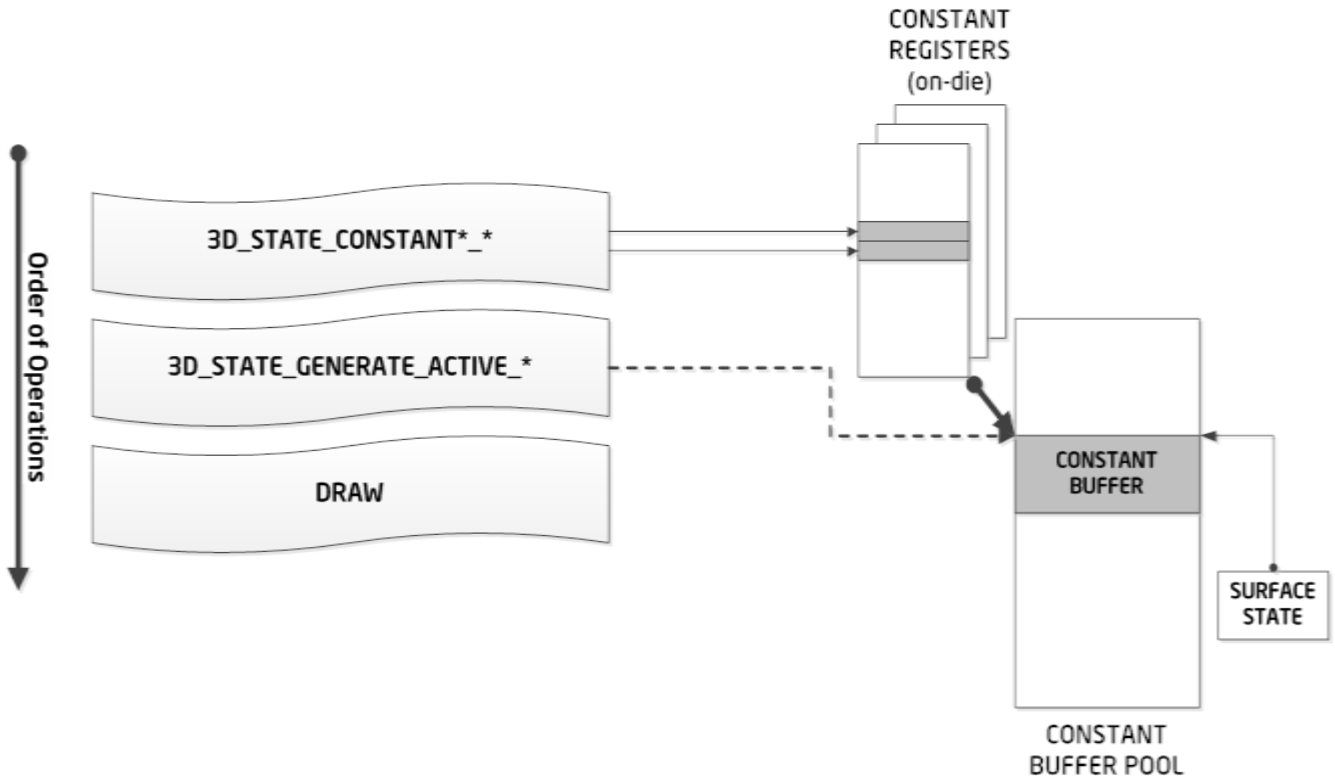

## Gather Push Constants Generation

## Constant Buffer Generation (not DX9)

The constant model used is a set of registers that the application can incrementally update. The hardware requires a constant buffer which lives until the last shader using that buffer retires. To offload the driver the 3D_STATE_CONSTANT*_* commands are used. The constant registers can be either floating, integer or Boolean (signified by the commands CONSTANTF, CONSTANTI, CONSTANTB, respectively). The option determines the fixed function for the constants (VS, GS, DS, HS, PS).

When all edits to the constant registers have been completed, the 3D_STATE_GENERATE_ACTIVE_* command is used to write out a constant buffer to the Constant Buffer Pool. These buffers are fixed at 8Kbytes. The software is required to provide a surface state object that points to the constant buffer created.

### Constant Buffer Generation




## Commands Actions in the RS

The tables below show all 3D commands processed by the RS. In the following tables, "STOP" indicates that the RS waits for all engines to complete operations AND invalidates all command data currently in the command FIFO. "BLOCK" indicates that the RS waits for all engines to complete operation, stops further command parsing, but retains data in the command FIFO.

### Table: MI Commands Processing in the RS

| Opcode (28:23) | Command | RS Handing – No Perf | RS Handling Perf | Comments |
|---|---|---|---|---|
| 03h | MI_WAIT_FOR_EVENT | STOP | BLOCK | |
| 05h | MI_ARB_CHECK | STOP | STOP | |
| 06h | MI_RS_CONTROL | STOP | STOP | |
| 0Ah | MI_BATCH_BUFFER_END | STOP | STOP | |
| 16h | MI_SEMAPHORE_MBOX | STOP | BLOCK | |
| 18h | MI_SET_CONTEXT | STOP | STOP | |
| 1Ah | MI_RS_CONTEXT | STOP | STOP | |
| 31h | MI_BATCH_BUFFER_START | STOP | STOP | |
| 36h | MI_CONDITIONAL_BATCH_BUFFER_END | STOP | STOP | |

### Table: Other Commands Processed in the RS

| Pipeline Type (28:27) | Opcode (26:24) | Sub Opcode (23:16) | Command | RS Handling (no perf) | RS Handling (perf) | Comments |
|---|---|---|---|---|---|---|
| 0h | 1h | 01h | STATE_BASE_ADDRESS | RS LATCH | RS LATCH | RSunit updates the state base address if parsed |
| 1h | 1h | 04h | PIPELINE_SELECT | STOP | STOP | Stop only if 3D is not selected |
| 3h | 0h | 03h | Reserved | | | |
| 3h | 0h | 04h | 3DSTATE_CLEAR_PARAMS [HSW] | | | Refer to 3D Pipeline |
| 3h | 0h | 05h | 3DSTATE_DEPTH_BUFFER [HSW] | | | Refer to 3D Pipeline |
| 3h | 0h | 06h | Reserved | | | |
| 3h | 0h | 06h | 3DSTATE_STENCIL_BUFFER [HSW] | | | Refer to 3D Pipeline |
| 3h | 0h | 07h | Reserved | | | |
| 3h | 0h | 07h | 3DSTATE_HIER_DEPTH_BUFFER [HSW] | | | Refer to 3D Pipeline |
| 3h | 0h | 08h | 3DSTATE_VERTEX_BUFFERS | | | Refer to Vertex Fetch |
| 3h | 0h | 09h | 3DSTATE_VERTEX_ELEMENTS | | | Refer to Vertex Fetch |
| 3h | 0h | 0Ah | 3DSTATE_INDEX_BUFFER | | | Refer to Vertex Fetch |
| 3h | 0h | 0Bh | 3DSTATE_VF_STATISTICS | | | Refer to Vertex Fetch |
| 3h | 0h | 0Ch | Reserved | | | |
| 3h | 0h | 0Dh | 3DSTATE_VIEWPORT_STATE_POINTERS [HSW] | | | Refer to 3D Pipeline |
| 3h | 0h | 10h | 3DSTATE_VS [HSW] | | | Refer to Vertex Shader |
| 3h | 0h | 11h | 3DSTATE_GS [HSW] | | | Refer to Geometry Shader |
| 3h | 0h | 12h | 3DSTATE_CLIP [HSW] | | | Refer to Clipper |
| 3h | 0h | 13h | 3DSTATE_SF [HSW] | | | Refer to Strips and Fans |
| 3h | 0h | 14h | 3DSTATE_WM [HSW] | | | Refer to Windower |
| 3h | 0h | 15h | 3DSTATE_CONSTANT_VS [HSW] | | | Refer to Vertex Shader |
| 3h | 0h | 16h | 3DSTATE_CONSTANT_GS [HSW] | | | Refer to Geometry Shader |
| 3h | 0h | 17h | 3DSTATE_CONSTANT_PS [HSW] | | | Refer to Windower |
| 3h | 0h | 18h | 3DSTATE_SAMPLE_MASK [HSW] | | | Refer to Windower |
| 3h | 0h | 19h | 3DSTATE_CONSTANT_HS [HSW] | | | Refer to Hull Shader |

| Pipeline Type (28:27) | Opcode (26:24) | Sub Opcode (23:16) | Command | RS Handling (no perf) | RS Handling (perf) | Comments |
|---|---|---|---|---|---|---|
| 3h | 0h | 1Ah | 3DSTATE_CONSTANT_DS [HSW] | | | Refer to Domain Shader |
| 3h | 0h | 1Bh | 3DSTATE_HS [HSW] | | | Refer to Hull Shader |
| 3h | 0h | 1Ch | 3DSTATE_TE [HSW] | | | Refer to Tesselator |
| 3h | 0h | 1Dh | 3DSTATE_DS [HSW] | | | Refer to Domain Shader |
| 3h | 0h | 1Eh | 3DSTATE_STREAMOUT [HSW] | | | Refer to HW Streamout |
| 3h | 0h | 1Fh | 3DSTATE_SBE [HSW] | | | Refer to Setup |
| 3h | 0h | 20h | 3DSTATE_PS [HSW] | | | Refer to Pixel Shader |
| 3h | 0h | 21h | Reserved | | | |
| 3h | 0h | 22h | 3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP [HSW] | | | Refer to Strips & Fans |
| 3h | 0h | 23h | 3DSTATE_VIEWPORT_STATE_POINTERS_CC [HSW] | | | Refer to Windower |
| 3h | 0h | 24h | 3DSTATE_BLEND_STATE_POINTERS [HSW] | | | Refer to Pixel Shader |
| 3h | 0h | 25h | 3DSTATE_DEPTH_STENCIL_STATE_POINTERS [HSW] | | | Refer to Pixel Shader |
| 3h | 0h | 26h | 3DSTATE_BINDING_TABLE_POINTERS_VS [HSW] | Generate BT if HW BT enabled | Generate BT if HW BT enabled | |
| 3h | 0h | 27h | 3DSTATE_BINDING_TABLE_POINTERS_HS [HSW] | Generate BT if HW BT enabled | Generate BT if HW BT enabled | |
| 3h | 0h | 28h | 3DSTATE_BINDING_TABLE_POINTERS_DS [HSW] | Generate BT if HW BT enabled | Generate BT if HW BT enabled | |
| 3h | 0h | 29h | 3DSTATE_BINDING_TABLE_POINTERS_GS [HSW] | Generate BT if HW BT enabled | Generate BT if HW BT enabled | |
| 3h | 0h | 2Ah | 3DSTATE_BINDING_TABLE_POINTERS_PS [HSW] | Generate BT if HW BT enabled | Generate BT if HW BT enabled | |
| 3h | 0h | 2Fh | Reserved | | | |
| 3h | 0h | 30h | 3DSTATE_URB_VS [HSW] | Execute | Execute | |
| 3h | 0h | 31h | 3DSTATE_URB_HS [HSW] | Execute | Execute | |
| 3h | 0h | 32h | 3DSTATE_URB_DS [HSW] | Execute | Execute | |

| Pipeline Type (28:27) | Opcode (26:24) | Sub Opcode (23:16) | Command | RS Handling (no perf) | RS Handling (perf) | Comments |
|---|---|---|---|---|---|---|
| 3h | 0h | 33h | 3DSTATE_URB_GS [HSW] | Execute | Execute | |
| 3h | 0h | 34h | 3DSTATE_GATHER_VS [HSW] | Execute | Execute | |
| 3h | 0h | 35h | 3DSTATE_GATHER_GS [HSW] | Execute | Execute | |
| 3h | 0h | 36h | 3DSTATE_GATHER_HS [HSW] | Execute | Execute | |
| 3h | 0h | 37h | 3DSTATE_GATHER_DS [HSW] | Execute | Execute | |
| 3h | 0h | 38h | 3DSTATE_GATHER_PS [HSW] | Execute | Execute | |
| 3h | 0h | 39h | 3DSTATE_CONSTANTF_VS [HSW] | Execute | Execute | |
| 3h | 0h | 3Ah | 3DSTATE_CONSTANTF_PS [HSW] | Execute | Execute | |
| 3h | 0h | 3Bh | 3DSTATE_CONSTANTI_VS [HSW] | Execute | Execute | |
| 3h | 0h | 3Ch | 3DSTATE_CONSTANTI_PS [HSW] | Execute | Execute | |
| 3h | 0h | 3Dh | 3DSTATE_CONSTANTB_VS [HSW] | Execute | Execute | |
| 3h | 0h | 3Eh | 3DSTATE_CONSTANTB_PS [HSW] | Execute | Execute | |
| 3h | 0h | 3Fh | 3DSTATE_LOCAL_VALID_VS [HSW] | Execute | Execute | |
| 3h | 0h | 40h | 3DSTATE_LOCAL_VALID_PS [HSW] | Execute | Execute | |
| 3h | 0h | 41h | 3DSTATE_GENERATE_ACTIVE_VS [HSW] | Execute | Execute | |
| 3h | 0h | 42h | 3DSTATE_GENERATE_ACTIVE_PS [HSW] | Execute | Execute | |
| 3h | 0h | 43h | 3DSTATE_BINDING_TABLE_EDIT_VS (DevHSW+) | | | Refer to Vertex Shader |
| 3h | 0h | 44h | 3DSTATE_BINDING_TABLE_EDIT_GS (DevHSW+) | | | Refer to Vertex Shader |
| 3h | 0h | 45h | 3DSTATE_BINDING_TABLE_EDIT_HS (DevHSW+) | | | Refer to Vertex Shader |
| 3h | 0h | 46h | 3DSTATE_BINDING_TABLE_EDIT_DS (DevHSW+) | | | Refer to Vertex Shader |
| 3h | 0h | 47h | 3DSTATE_BINDING_TABLE_EDIT_PS (DevHSW+) | | | Refer to Vertex Shader |
| 3h | 0h | 4Ch | 3DSTATE_WM_CHROMA_KEY | | | |
| 3h | 0h | 4Dh | 3DSTATE_PS_BLEND | | | |
| 3h | 0h | 4Eh | 3DSTATE_WM_DEPTH_STENCIL | | | |
| 3h | 0h | 4Fh | 3DSTATE_PS_EXTRA | | | |
| 3h | 0h | 50h | 3DSTATE_RASTER | | | |
| 3h | 0h | 51h | 3DSTATE_SBE_SWIZ | | | |
| 3h | 0h | 52h | 3DSTATE_WM_HZ_OP | | | |

| Pipeline Type (28:27) | Opcode (26:24) | Sub Opcode (23:16) | Command | RS Handling (no perf) | RS Handling (perf) | Comments |
|---|---|---|---|---|---|---|
| 3h | 0h | 53h | 3DSTATE_INT (internally generated state) | | | |
| 3h | 0h | 57h | 3DSTATE_DX9_CONSTANTF_HS [HSW] | | | |
| 3h | 0h | 58h | 3DSTATE_DX9_CONSTANTI_HS [HSW] | | | |
| 3h | 0h | 59h | 3DSTATE_DX9_CONSTANTB_HS [HSW] | | | |
| 3h | 0h | 5ah | 3DSTATE_DX9_LOCAL_VALID_HS [HSW] | | | |
| 3h | 0h | 5bh | 3DSTATE_DX9_GENERATE_ACTIVE_HS [HSW] | | | |
| 3h | 0h | 5ch | 3DSTATE_DX9_CONSTANTF_DS [HSW] | | | |
| 3h | 0h | 5dh | 3DSTATE_DX9_CONSTANTI_DS [HSW] | | | |
| 3h | 0h | 5eh | 3DSTATE_DX9_CONSTANTB_DS [HSW] | | | |
| 3h | 0h | 5fh | 3DSTATE_DX9_LOCAL_VALID_DS [HSW] | | | |
| 3h | 0h | 60h | 3DSTATE_DX9_GENERATE_ACTIVE_DS [HSW] | | | |
| 3h | 0h | 61h | 3DSTATE_DX9_CONSTANTF_GS [HSW] | | | |
| 3h | 0h | 62h | 3DSTATE_DX9_CONSTANTI_GS [HSW] | | | |
| 3h | 0h | 63h | 3DSTATE_DX9_CONSTANTB_GS [HSW] | | | |
| 3h | 0h | 64h | 3DSTATE_DX9_LOCAL_VALID_GS [HSW] | | | |
| 3h | 0h | 65h | 3DSTATE_DX9_GENERATE_ACTIVE_GS [HSW] | | | |
| 3h | 0h | 66h | 3DSTATE_GATHER_GPGPU | | | |
| 3h | 0h | 67h-FFh | Reserved | | | |
| 3h | 1h | 00h | 3DSTATE_DRAWING_RECTANGLE | | | |
| 3h | 1h | 02h | 3DSTATE_SAMPLER_PALETTE_LOAD0 | | | |
| 3h | 1h | 03h | Reserved | | | |
| 3h | 1h | 04h | 3DSTATE_CHROMA_KEY | | | |
| 3h | 1h | 05h | Reserved [HSW] | | | |
| 3h | 1h | 06h | 3DSTATE_POLY_STIPPLE_OFFSET | | | |
| 3h | 1h | 07h | 3DSTATE_POLY_STIPPLE_PATTERN | | | |
| 3h | 1h | 08h | 3DSTATE_LINE_STIPPLE | | | |
| 3h | 1h | 0Ah | 3DSTATE_AA_LINE_PARAMS | | | |
| 3h | 1h | 0Bh | 3DSTATE_GS_SVB_INDEX | | | |
| 3h | 1h | 0Ch | 3DSTATE_SAMPLER_PALETTE_LOAD1 | | | |

| Pipeline Type (28:27) | Opcode (26:24) | Sub Opcode (23:16) | Command | RS Handling (no perf) | RS Handling (perf) | Comments |
|---|---|---|---|---|---|---|
| 3h | 1h | 0Dh | 3DSTATE_MULTISAMPLE [HSW] | | | |
| 3h | 1h | 0Eh | 3DSTATE_STENCIL_BUFFER | | | |
| 3h | 1h | 0Fh | 3DSTATE_HIER_DEPTH_BUFFER | | | |
| 3h | 1h | 10h | 3DSTATE_CLEAR_PARAMS | | | |
| 3h | 1h | 11h | 3DSTATE_MONOFILTER_SIZE | | | |
| 3h | 1h | 12h | 3DSTATE_PUSH_CONSTANT_ALLOC_VS [HSW] | | | |
| 3h | 1h | 13h | 3DSTATE_PUSH_CONSTANT_ALLOC_HS [HSW] | | | |
| 3h | 1h | 14h | 3DSTATE_PUSH_CONSTANT_ALLOC_DS [HSW] | | | |
| 3h | 1h | 15h | 3DSTATE_PUSH_CONSTANT_ALLOC_GS [HSW] | | | |
| 3h | 1h | 16h | 3DSTATE_PUSH_CONSTANT_ALLOC_PS [HSW] | | | |
| 3h | 1h | 17h | 3DSTATE_SO_DECL_LIST | | | |
| 3h | 1h | 18h | 3DSTATE_SO_BUFFER | | | |
| 3h | 1h | 19h | 3DSTATE_BINDING_TABLE_POOL_ALLOC [HSW] | | | |
| 3h | 1h | 1Ah | 3DSTATE_GATHER_POOL_ALLOC [HSW] | | | |
| 3h | 1h | 1Bh | 3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC [HSW] | | | |
| 3h | 1h | 1Ch | 3DSTATE_SAMPLE_PATTERN | | | |
| 3h | 1h | 1Dh-FFh | Reserved | | | |
| 3h | 1h | 19h | 3DSTATE_BINDING_TABLE_POOL_ALLOC [HSW] | Execute | Execute | |
| 3h | 1h | 1Ah | 3DSTATE_GATHER_POOL_ALLOC [HSW] | Execute | Execute | |
| 3h | 1h | 1Bh | 3DSTATE_CONSTANT_BUFFER_POOL_ALLOC [HSW] | Execute | Execute | |
| 3h | 1h | 1Ch | 3DSTATE_SAMPLE_PATTERN | | | |
| 3h | 1h | 1Dh-FFh | Reserved | | | |
| 3h | 2h | 00h | PIPE_CONTROL | | | |

| Pipeline Type (28:27) | Opcode (26:24) | Sub Opcode (23:16) | Command | RS Handling (no perf) | RS Handling (perf) | Comments |
|---|---|---|---|---|---|---|
| 3h | 2h | 01h-FFh | Reserved | | | |
| 3h | 3h | 00h | 3DPRIMITIVE | Sync | Sync | 3DPRIMITIVE command is unique in that it tells the engines to send fence cycles, but does not stop RSunit (not a sync point) |

## Resource Streamer Programming Guidelines

This section describes RS activities and assumptions that are required for programming.

## RS Interactions with the 3D Command Streamer

Because the Resource Streamer is processing ahead of the Command Streamer, many of the commands interpreted by the RS are a signal to stop further processing. In these cases, the RS completes pending activity, and waits for an indication from the Command Streamer to start again.

The specific cases that the CS commands the RS to continue are:

- Batch Buffer command parsing
- Context save

## RS Interactions with Memory Requests

The RS is responsible for the generation of a number of memory requests. These are:

- Make batch buffer read requests (when address is supplied from the CS)
- Make push constant gather read requests from the state base offset
- Make push constant gather write of packed data to the gather pool
- Fetch the gather buffer surface base address
- Write out the binding table pointer (BTP)
- Saving BTP, constant buffer and gather constant context data to an offset into the context image
- Writing out constant data

As is the case in all memory accesses, the read requests from the RS can be freely reordered, and may be returned in any order by the hardware. The RS will consume the cycles, and present the "software" order transparently.

When accessing the same address, a write operation followed by the read will return the written data. Writes to non-overlapping addresses may be freely reordered as well. Fencing is used to make certain all writes up to the fence have completed.

## Fundamental Programming and Operational Assumptions

The following assumptions have been made in the RS, and these are useful limitations to the programming.

- The CS can never send a request to a busy RS.  The RS will have foreseen the situation, and stopped it operations prior to the CS action.
- Surface base address will never be changed while in a batch buffer
- Push constant data is expected to be 128-bit aligned
- The GATHER command should have Constant Buffer valid bits set for any indices used in the command

## Non-Operational Activities

There are no specific events or performance counters for the RS.