Intel® Iris® Xᵉ MAX Graphics Open Source

Programmer's Reference Manual

For the 2020 Discrete GPU formerly named "DG1"

Volume 13: General Assets

February 2021, Revision 1.0

Notices and Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks.

Customer is responsible for safety of the overall system, including compliance with applicable safety-related requirements or standards.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

The products described 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.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

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.

Table of Contents

General Assets

This is the General Assets section.

Discrete Graphics Memory Map Information

Intel graphics subsystem need certain *reserved* area in memory for various use case. Memory can be system memory for Integrated GFX or Local memory for discrete GFX. This chapter provides information on *reserved* area and provide general guideline for recommended size & location of memory.

During boot, for discrete graphics, a secure uController (like GSC) will program the memory map. Some of reserved area is also setup by drivers. Region size and placement is passed to graphics hardware and software via defined MMIO register locations. The general category of this reserved section of memory are:

- **D**ata **S**tolen **M**emory (DSM): Stolen memory used typically for pre-OS Display Frame buffers like in GOP/VBIOS and within Driver as general GFX memory. Note: Driver typically reuses this area for Frame Buffer compression (FBC)

- **W**rite-**O**nce-**P**rotected-**C**ontent-**M**emory (WOPCM): Used for storing Authenticated FW like RC6 context, etc.

- **G**lobal Translation Table(GTT) **S**tolen **M**emory (GSM): Location of the Global Graphics Translation Table.

- **Flat C**ompression **C**ontrol **S**urface (Flat CCS): Compression table within Local memory.

- **L**ocal **M**emory **T**ranslation **T**able (LMTT): When virtualized, location of the local memory translation table.

PCIe MMIO Memory BARs :

- LMEMBAR :
    - o 64 bit prefetchable BAR
    - o The driver requests the OS to map the LMEMBAR range as memory type write-combining (ie. USWC) .. for performance reasons.
- GTTMMADR :
    - o 64 bit prefetchable BAR
    - o Uncacheable (ie. UC).
    - o GTTMMADR(Reg) : R/W will be 32 bit aligned and 32 bit (or less) in length. Driver requirement similar to prior art.
    - o GTTMMADR(GTT) : R/W will be 64 bit aligned and 64 bits (or less) in length. Driver requirement similar to prior art.

MMIO

This is the MMIO chapter.

Force Wake and Steering Table

Here is the Force Wake and Steering Table.

MMIO Range Start	MMIO Range End	# Bytes	Wake Target	Replicated / Multicast ?	Replication Group Type	Inst. Count	Steering
00000000	00000AFF	2816					
00000B00	00000BFF	256	AON	Yes	SQIDI	2	subsliceid[0..1]
00000C00	00000DFF	512	AON	No	-	1	-
00000E00	00000FFF	512	AON	No	-	1	-
00001000	00001FFF	4096	AON	Yes	SQIDI	2	subsliceid[0..1]
00002000	000026FF	1792	RENDER	No	-	1	-
00002700	000027FF	256	GT	No	-	1	-
00002800	00002AFF	768	RENDER	No	-	1	-
00002B00	00002FFF	1280	GT	No	-	1	-
00003000	00003FFF	4096	RENDER	No	-	1	-
00004000	000041FF	512	GT	No	-	1	-
00004200	000043FF	512	GT	No	-	1	-
00004400	000048FF	1280	GT	No	-	1	-
00004900	00004FFF	1792					
00005000	000051FF	512					
00005200	000052FF	256	RENDER	No	-	1	-
00005300	000053FF	256	RENDER	No	-	1	-
00005400	000054FF	256					
00005500	00005FFF	2816	RENDER	No	-	1	-
00006000	00006FFF	4096	RENDER	No	-	1	-
00007000	00007FFF	4096	RENDER	No	-	1	-
00008000	000080FF	256	GT	No	-	1	-
00008100	0000813F	64	GT	No	-	1	-
00008140	0000814F	16	RENDER	No	-	1	-
00008150	0000815F	16	RENDER	Yes	DSS	6	subsliceid[0..5]
00008160	0000817F	32					
00008180	000081FF	128	AON	No	-	1	-
00008200	000082FF	256	GT	No	-	1	-
00008300	000084FF	512	RENDER	No	-	1	-
00008500	000085FF	256	GT	No	-	1	-

00008600	000086FF	256	GT	No	-	1	-
00008700	000087FF	256	GT	Yes	SQIDI	2	subsliceid[0..1]
00008800	00008FFF	2048					
00009000	000093FF	1024	GT	No	-	1	-
00009400	0000947F	128	GT	No	-	1	-
00009480	000094CF	80					
000094D0	0000951F	80	RENDER	No	-	1	-
00009520	0000955F	64	RENDER	Yes	DSS	6	subsliceid[0..5]
00009560	000095FF	160	AON	No	-	1	-
00009600	000097FF	512					
00009800	00009FFF	2048	GT	No	-	1	-
0000A000	0000AFFF	4096	GT	No	-	1	-
0000B000	0000B0FF	256	RENDER	No	-	1	-
0000B100	0000B3FF	768	RENDER	Yes	L3BANK	8	subsliceid[0..7]
0000B400	0000B47F	128	GT	No	-	1	-
0000B480	0000BFFF	2944					
0000C000	0000C7FF	2048	GT	No	-	1	-
0000C800	0000CFFF	2048	GT	No	-	1	-
0000D000	0000D3FF	1024	AON	No	-	1	-
0000D400	0000D7FF	1024	AON	No	-	1	-
0000D800	0000D8FF	256	RENDER	No	-	1	-
0000D900	0000DBFF	768	GT	No	-	1	-
0000DC00	0000DDFF	512	RENDER	No	-	1	-
0000DE00	0000DE7F	128					
0000DE80	0000DEFF	128	RENDER	Yes	DSS	6	subsliceid[0..5]
0000DF00	0000DFFF	256	RENDER	Yes	DSS	6	subsliceid[0..5]
0000E000	0000E0FF	256	RENDER	Yes	DSS	6	subsliceid[0..5]
0000E100	0000E1FF	256	RENDER	Yes	DSS	6	subsliceid[0..5]
0000E200	0000E3FF	512	RENDER	Yes	DSS	6	subsliceid[0..5]
0000E400	0000E7FF	1024	RENDER	Yes	DSS	6	subsliceid[0..5]
0000E800	0000E8FF	256	RENDER	Yes	DSS	6	subsliceid[0..5]
0000E900	0000EFFF	1792					
0000F000	0000F0FF	256	GT	No	-	1	-
0000F100	0000FFFF	3840	GT	No	-	1	-
00010000	000147FF	18432					
00014800	00014FFF	2048	RENDER	No	-	1	-
00015000	00016DFF	7680					
00016E00	00016FFF	512	RENDER	No	-	1	-
00017000	00017FFF	4096	RENDER	No	-	1	-

00018000	00019FFF	8192	RENDER	No	-	1	-
0001A000	0001BFFF	8192	RENDER	No	-	1	-
0001C000	0001DFFF	8192					
0001E000	0001FFFF	8192					
00020000	00020FFF	4096	VD0	No	-	1	-
00021000	00021FFF	4096	VD2	No	-	1	-
00022000	00022FFF	4096	GT	No	-	1	-
00023000	00023FFF	4096	GT	No	-	1	-
00024000	0002407F	128	AON	No	-	1	-
00024080	0002417F	256					
00024180	000241FF	128	GT	No	-	1	-
00024200	000249FF	2048					
00024A00	00024A7F	128	RENDER	Yes	DSS	6	subsliceid[0..5]
00024A80	000251FF	1920					
00025200	0002527F	128	GT	No	-	1	-
00025280	000252FF	128	GT	No	-	1	-
00025300	000255FF	768					
00025600	0002567F	128	VD0	No	-	1	-
00025680	000256FF	128	VD2	No	-	1	-
00025700	000259FF	768					
00025A00	00025A7F	128	VD0	No	-	1	-
00025A80	00025AFF	128	VD2	No	-	1	-
00025B00	00025FFF	1280					
00026000	00027FFF	8192					
00028000	0002FFFF	32768					
00030000	0003FFFF	65536	GT	No	-	1	-

MMIO Range Start	MMIO Range End	# Bytes	Wake Target	Replicated / Multicast ?	Replication Group Type	Inst. Count	Steering
001C0000	001C07FF	2048	VD0	No	-	1	-
001C0800	001C0FFF	2048	VD0	No	-	1	-
001C1000	001C1FFF	4096	VD0	No	-	1	-
001C2000	001C27FF	2048	VD0	No	-	1	-
001C2800	001C2AFF	768	VD0	No	-	1	-
001C2B00	001C2BFF	256	VD0	No	-	1	-
001C2C00	001C2CFF	256					
001C2D00	001C2DFF	256	VD0	No	-	1	-
001C2E00	001C3EFF	4352					
001C3F00	001C3FFF	256	VD0	No	-	1	-
001C4000	001C47FF	2048					
001C4800	001C4FFF	2048					
001C5000	001C5FFF	4096					
001C6000	001C67FF	2048					
001C6800	001C6AFF	768					
001C6B00	001C6BFF	256					
001C6C00	001C6CFF	256					
001C6D00	001C6DFF	256					
001C6E00	001C7EFF	4352					
001C7F00	001C7FFF	256					
001C8000	001C9FFF	8192	VE0	No	-	1	-
001CA000	001CA0FF	256	VE0	No	-	1	-
001CA100	001CBEFF	7680					
001CBF00	001CBFFF	256	VE0	No	-	1	-
001CC000	001CCFFF	4096	VD0	No	-	1	-
001CD000	001CDFFF	4096					
001CE000	001CEFFF	4096					
001CF000	001CFFFF	4096					
001D0000	001D07FF	2048	VD2	No	-	1	-
001D0800	001D0FFF	2048	VD2	No	-	1	-
001D1000	001D1FFF	4096	VD2	No	-	1	-
001D2000	001D27FF	2048	VD2	No	-	1	-
001D2800	001D2AFF	768	VD2	No	-	1	-
001D2B00	001D2BFF	256	VD2	No	-	1	-
001D2C00	001D2CFF	256					

001D2D00	001D2DFF	256	VD2	No	-	1	-
001D2E00	001D3EFF	4352					
001D3F00	001D3FFF	256	VD2	No	-	1	-
001D4000	001D47FF	2048					
001D4800	001D4FFF	2048					
001D5000	001D5FFF	4096					
001D6000	001D67FF	2048					
001D6800	001D6AFF	768					
001D6B00	001D6BFF	256					
001D6C00	001D6CFF	256					
001D6D00	001D6DFF	256					
001D6E00	001D7EFF	4352					
001D7F00	001D7FFF	256					
001D8000	001D9FFF	8192					
001DA000	001DA0FF	256					
001DA100	001DBEFF	7680					
001DBF00	001DBFFF	256					
001DC000	001DFFFF	16384					
001E0000	001E07FF	2048					
001E0800	001E0FFF	2048					
001E1000	001E1FFF	4096					
001E2000	001E27FF	2048					
001E2800	001E2AFF	768					
001E2B00	001E2BFF	256					
001E2C00	001E2CFF	256					
001E2D00	001E2DFF	256					
001E2E00	001E3EFF	4352					
001E3F00	001E3FFF	256					
001E4000	001E47FF	2048					
001E4800	001E4FFF	2048					
001E5000	001E5FFF	4096					
001E6000	001E67FF	2048					
001E6800	001E6AFF	768					
001E6B00	001E6BFF	256					
001E6C00	001E6CFF	256					
001E6D00	001E6DFF	256					
001E6E00	001E7EFF	4352					
001E7F00	001E7FFF	256					
001E8000	001E9FFF	8192					

001EA000	001EA0FF	256					
001EA100	001EBEFF	7680					
001EBF00	001EBFFF	256					
001EC000	001EFFFF	16384					
001F0000	001F07FF	2048					
001F0800	001F0FFF	2048					
001F1000	001F1FFF	4096					
001F2000	001F27FF	2048					
001F2800	001F2AFF	768					
001F2B00	001F2BFF	256					
001F2C00	001F2CFF	256					
001F2D00	001F2DFF	256					
001F2E00	001F3EFF	4352					
001F3F00	001F3FFF	256					
001F4000	001F47FF	2048					
001F4800	001F4FFF	2048					
001F5000	001F5FFF	4096					
001F6000	001F67FF	2048					
001F6800	001F6AFF	768					
001F6B00	001F6BFF	256					
001F6C00	001F6CFF	256					
001F6D00	001F6DFF	256					
001F6E00	001F7EFF	4352					
001F7F00	001F7FFF	256					
001F8000	001F9FFF	8192					
001FA000	001FA0FF	256					
001FA100	001FBEFF	7680					
001FBF00	001FBFFF	256					
001FC000	001FFFFF	16384					
00200000	0023FFFF	262144					

- The Steering Control Registers reside at the following locations:
    - MGSR access point (access initiated by agent outside of GT):

#	Steering Reg Addr	Description
1	0xFD0	Access steering towards MCFG endpoints only.
2	0xFD4	Access steering towards MDRB endpoints only
3	0xFD8	Access steering towards SF endpoints only
4	0xFDC	Access steering towards all other endpoints (all but above)

- CS access point:

#	Steering Reg Addr	Description
1	0x20CC	Access steering towards all GT endpoints

- **Note:** All Steering Control Registers contain the following fields:

Field	Description
multicast	1: Access will be multicast to all replicated endpoints: • *WRITE* op cycles go to all endpoint instances; sliceid[]/subsliceid[] fields ignored. • *READ* op cycles go to all endpoint instances, and responses are returned from all instances; The MsgCh selects single instance's response as the final read return, based on sliceid[]/subsliceid[] fields. 0: Access will be steered using sliceid[] and subsliceid[] fields below: • Both *WRITE* and *READ* cycles go to a single instance of an endpoint, based on sliceid[]/subsliceid[] steering. Default: 1 Note: The multicast field has no impact for a non-replicated target.
sliceid[]	Default: 0
subsliceid[]	Default: 0

-

- The following Replication Group Types exist for multicast MMIO endpoints:

Replication Group Type	Description / Notes
SQIDI	• 2 instances • subsliceid: 0..1 • all instances are always present.
DSS	• LP has max 6 DSS • subsliceid: 0..5 • Terminated/disabled when the corresponding dss_enable bit is '0'
L3BANK	• 8 instances • subsliceid 0..7 to access • Terminated/disabled when corresponding fuse_gt_l3disable bit is 'disable'

- Fuse reflections (how to tell when an endpoint is disabled):

Fuse	Register reflection
fuse_gt_dssen[5:0]	0x913C[5:0]
fuse_gt_l3_disable[3:0]	0x9118[7:0] (fuse is replicated into [3:0] and [7:4])

Note: MsgCh termination also occurs when the domains are powered down. (i.e., not necessarily because the domain is disabled/fused off.) If reading/writing the registers is needed, then force-wake of the domain is required. Force-wake is not required for shadow register accesses coming through MGSR.

- The following table captures the force-wake and corresponding acknowledgment register locations for the various domains:

Domain	Driver ForceWake Req	Driver ForceWake Ack	Comment
AON	NA	NA	Registers sit outside of the C6 boundary. No ForceWake required.
GT	0xA188	0x00130044	
Render	0xA278	0x0D84	
VDBOX0	0xA540	0x0D50	
VDBOX1	0xA544	0x0D54	
VDBOX2	0xA548	0x0D58	
VDBOX3	0xA54C	0x0D5C	As available per config
VDBOX4	0xA550	0x0D60	
VDBOX5	0xA554	0x0D64	
VDBOX6	0xA558	0x0D68	
VDBOX7	0xA55C	0x0D6C	
VEBOX0	0xA560	0x0D70	
VEBOX1	0xA564	0x0D74	
VEBOX2	0xA568	0x0D78	
VEBOX3	0xA56C	0x0D7C	

- Miscellaneous Notes:
- The MsgCh network has termination points, where cycles to endpoints that are disabled (fused-off, powered off, etc.) are gracefully completed. The termination node on the network will sink P cycles, and return dummy completions for NP cycles, on behalf of the disabled endpoints.
- Access requirements to registers that are part of GTMMADDR but not listed in the GT MMIO map table is defined elsewhere. This descriptions in this document only cover GT range (GT MMIO map xls.)

Slice Registers and Die Recovery

When slice 0 is disabled (for example, GT3 fused to GT2 with a slice 0 fault), any read to a slice-located MMIO register must be directed to slice 1, otherwise data of '0' will be returned. This applies to SRM cycles from any command streamer.

MMIO Range Start	MMIO Range End	Unit Description
00005500	00005FFF	WMBE
00007000	00007FFF	SVL
00009400	000097FF	CP unit reg. file - Copy in Slice Common (in all slices)
0000B000	0000B0FF	L3 unique status registers for each slice (unicast per GT).
0000B100	0000B3FF	L3 bank config space (multicast copy per bank and slice)
0000E000	0000E0FF	DM
0000E100	0000E1FF	SC
0000E200	0000E3FF	GWL (inst. 0)
0000E200	0000E3FF	GWL (inst. 1)
0000E200	0000E3FF	GWL (inst. 2)
0000E400	0000E7FF	TDL

SW Virtualization Reserved MMIO range

The MMIO address range from 0x178000 thru 0x178FFF is reserved for communication between a VMM and the GPU Driver executing on a Virtual Machine.

HW does not actually implement anything within this range. Instead, in a SW Virtualized environment, if a VM driver issues a read to this MMIO address range, the VMM will trap that access, and provide whatever data it wishes to pass to the VM driver. In a non-SW-Virtualizated environment (including an SR-IOV Virtualized environment), reads will return zeros, like any other unimplemented MMIO address. Writes to this range are always ignored.

It is important that no "real" HW MMIO register be defined within this range, as it would be inaccessible in a SW-virtualized environment.

Register Address Maps

Graphics Register Address Map

This chapter provides address maps of the graphics controllers I/O and memory-mapped registers. Individual register bit field descriptions are provided in the following chapters. PCI configuration address maps and register bit descriptions are provided in the following chapter.

VGA and Extended VGA Register Map

For I/O locations, the value in the address column represents the register I/O address. For memory mapped locations, this address is an offset from the base address programmed in the MMADR register.

VGA and Extended VGA I/O and Memory Register Map

Address	Register Name (Read)	Register Name (Write)
2D Registers		
3B0h-3B3h	Reserved	Reserved
3B4h	VGA CRTC Index (CRX) (monochrome)	VGA CRTC Index (CRX) (monochrome)
3B5h	VGA CRTC Data (monochrome)	VGA CRTC Data (monochrome)
3B6h-3B9h	Reserved	Reserved
3Bah	VGA Status Register (ST01)	VGA Feature Control Register (FCR)
3BBh-3BFh	Reserved	Reserved
3C0h	VGA Attribute Controller Index (ARX)	VGA Attribute Controller Index (ARX)/ VGA Attribute Controller Data (alternating writes select ARX or write ARxx Data)
3C1h	VGA Attribute Controller Data (read ARxx data)	Reserved
3C2h	VGA Feature Read Register (ST00)	VGA Miscellaneous Output Register (MSR)
3C3h	Reserved	Reserved
3C4h	VGA Sequencer Index (SRX)	VGA Sequencer Index (SRX)
3C5h	VGA Sequencer Data (SRxx)	VGA Sequencer Data (SRxx)
3C6h	VGA Color Palette Mask (DACMASK)	VGA Color Palette Mask (DACMASK)
3C7h	VGA Color Palette State (DACSTATE)	VGA Color Palette Read Mode Index (DACRX)
3C8h	VGA Color Palette Write Mode Index (DACWX)	VGA Color Palette Write Mode Index (DACWX)
3C9h	VGA Color Palette Data (DACDATA)	VGA Color Palette Data (DACDATA)
3CAh	VGA Feature Control Register (FCR)	Reserved
3CBh	Reserved	Reserved
3CCh	VGA Miscellaneous Output Register (MSR)	Reserved
3CDh	Reserved	Reserved
3CEh	VGA Graphics Controller Index (GRX)	VGA Graphics Controller Index (GRX)
3CFh	VGA Graphics Controller Data (GRxx)	VGA Graphics Controller Data (GRxx)
3D0h-3D1h	Reserved	Reserved
2D Registers		

Address	Register Name (Read)	Register Name (Write)
3D4h	VGA CRTC Index (CRX)	VGA CRTC Index (CRX)
3D5h	VGA CRTC Data (CRxx)	VGA CRTC Data (CRxx)
System Configuration Registers		
3D6h	GFX/2D Configurations Extensions Index (XRX)	GFX/2D Configurations Extensions Index (XRX)
3D7h	GFX/2D Configurations Extensions Data (XRxx)	GFX/2D Configurations Extensions Data (XRxx)
2D Registers		
3D8h-3D9h	Reserved	Reserved
3DAh	VGA Status Register (ST01)	VGA Feature Control Register (FCR)
3DBh-3DFh	Reserved	Reserved

Indirect VGA and Extended VGA Register Indices

The registers listed in this section are indirectly accessed by programming an index value into the appropriate SRX, GRX, ARX, or CRX register. The index and data register address locations are listed in the previous section. Additional details concerning the indirect access mechanism are provided in the *VGA and Extended VGA Register Description* Chapter (see SRxx, GRxx, ARxx or CRxx sections).

2D Sequence Registers (3C4h / 3C5h)

Index	Sym	Description
00h	SR00	Sequencer Reset
01h	SR01	Clocking Mode
02h	SR02	Plane / Map Mask
03h	SR03	Character Font
04h	SR04	Memory Mode
07h	SR07	Horizontal Character Counter Reset

2D Graphics Controller Registers (3CEh / 3CFh)

Index	Sym	Register Name
00h	GR00	Set / Reset
01h	GR01	Enable Set / Reset
02h	GR02	Color Compare
03h	GR03	Data Rotate
04h	GR04	Read Plane Select
05h	GR05	Graphics Mode
06h	GR06	Miscellaneous
07h	GR07	Color Don't Care

Index	Sym	Register Name
08h	GR08	Bit Mask
10h	GR10	Address Mapping
11h	GR11	Page Selector
18h	GR18	Software Flags

2D Attribute Controller Registers (3C0h / 3C1h)

Index	Sym	Register Name
00h	AR00	Palette Register 0
01h	AR01	Palette Register 1
02h	AR02	Palette Register 2
03h	AR03	Palette Register 3
04h	AR04	Palette Register 4
05h	AR05	Palette Register 5
06h	AR06	Palette Register 6
07h	AR07	Palette Register 7
08h	AR08	Palette Register 8
09h	AR09	Palette Register 9
0Ah	AR0A	Palette Register A
0Bh	AR0B	Palette Register B
0Ch	AR0C	Palette Register C
0Dh	AR0D	Palette Register D
0Eh	AR0E	Palette Register E
0Fh	AR0F	Palette Register F
10h	AR10	Mode Control
11h	AR11	Overscan Color
12h	AR12	Memory Plane Enable
13h	AR13	Horizontal Pixel Panning
14h	AR14	Color Select

2D CRT Controller Registers (3B4h / 3D4h / 3B5h / 3D5h)

Index	Sym	Register Name
00h	CR00	Horizontal Total
01h	CR01	Horizontal Display Enable End
02h	CR02	Horizontal Blanking Start
03h	CR03	Horizontal Blanking End
04h	CR04	Horizontal Sync Start
05h	CR05	Horizontal Sync End
06h	CR06	Vertical Total

Index	Sym	Register Name
07h	CR07	Overflow
08h	CR08	Preset Row Scan
09h	CR09	Maximum Scan Line
0Ah	CR0A	Text Cursor Start
0Bh	CR0B	Text Cursor End
0Ch	CR0C	Start Address High
0Dh	CR0D	Start Address Low
0Eh	CR0E	Text Cursor Location High
0Fh	CR0F	Text Cursor Location Low
10h	CR10	Vertical Sync Start
11h	CR11	Vertical Sync End
12h	CR12	Vertical Display Enable End
13h	CR13	Offset
14h	CR14	Underline Location
15h	CR15	Vertical Blanking Start
16h	CR16	Vertical Blanking End
17h	CR17	CRT Mode
18h	CR18	Line Compare
22h	CR22	Memory Read Latch Data

Command Stream Programming

This is the Command Stream Programming section.

Introduction

Command Streamer is the primary interface to the various engines that are part of the graphics hardware.

The graphics HW consists of multiple parallel engines that can execute different kinds of workloads. E.g. Render engine for 3D and GPGPU tasks, Video Decode engine, Video Enhancement Engine and Blitter engine.

Some product SKU's have multiple instances of an engine (e.g. 2 Video Decode engines).

As shown in figure 1, each of these engines have their own Command Streamer that is responsible for processing the commands in the workload and enabling execution of the task.

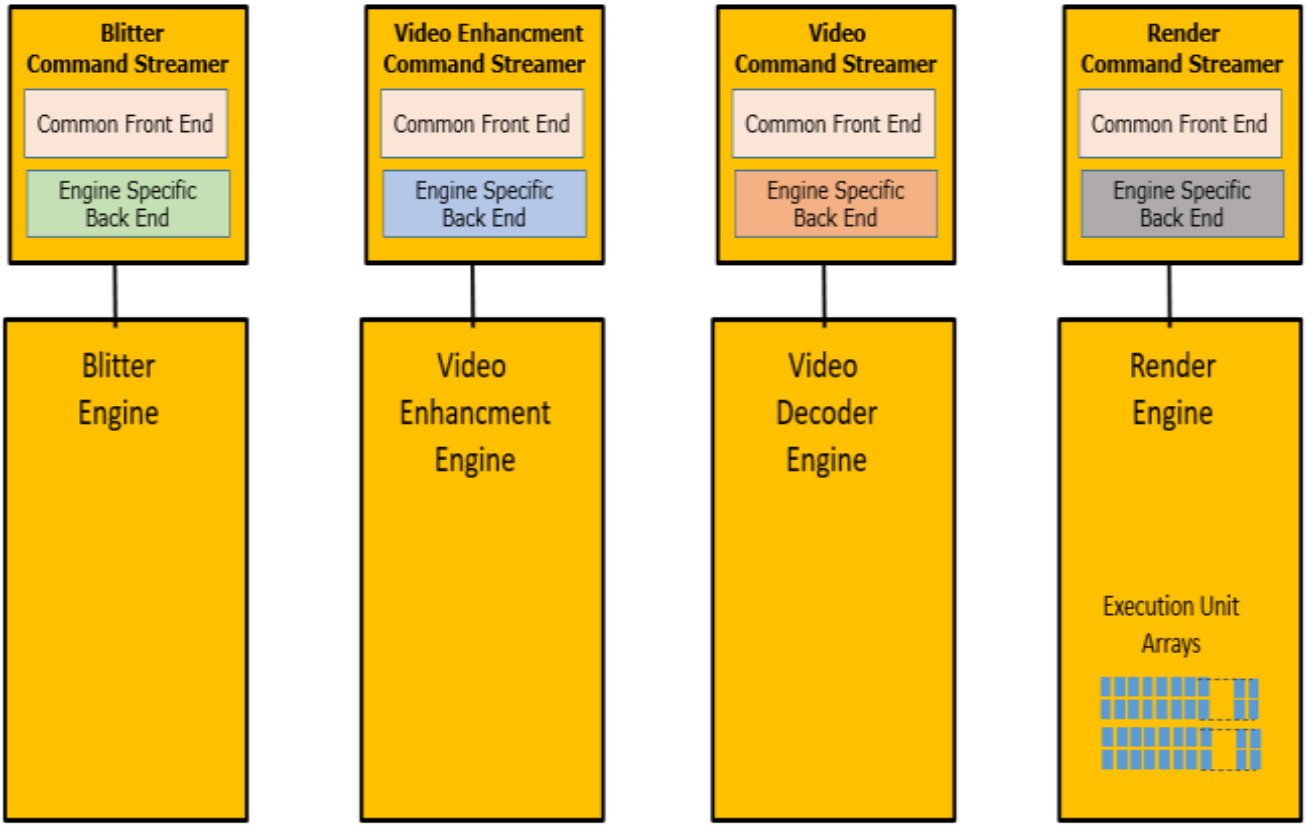


Figure 1: High level view of Command Streamer

As shown in the figure, the command streamer is comprised of a Common Front end and an engine specific backend.

The common front end allows each engine to provide a uniform software interface (e.g. infrastructure for submission of commands, synchronization, etc.).

The back ends handle the engine specific commands and the protocols required to control the execution of the underlying engine.

Workload Submission and Execution Status

This section describes the interface to submit work and obtain status

Scheduling and Execlists

Execution-List provides a HW-SW interface mechanism to schedule context as a fundamental unit of submission to GFX-device for execution. GFX-device has multiple engines (Render, Blitter, Video, Video Enhancement) with each of them having an execution list for context submission. At any given time, all engines could be concurrently running different contexts.

A context is identified with a unique identifier called Context ID. Each context is associated with an address space for memory accesses and is assigned a unique ring buffer for command submission.

SW submits workload for a context by programming commands into its assigned ring buffer prior to submitting context to HW (engine) for execution.

Context State:

Each context programs the engine state according to its workload requirements. All the hardware state variables of an engine required to execute a context is called context state. Each context has its own context state. Context state gets programmed on execution of commands from the context ring buffer. All the contexts designated to run on an engine have the same context format, however the values may differ based on the individual state programming.

Logical Context Address:

Each context is assigned a Logical Context Address to which the context state is saved by the engine on a context getting switched out from execution. Similarly, engine restores the context state from the logical context address of a context on getting switched in for execution.

Logical context address is an absolute graphics virtual address in global virtual memory. Context state save/restore mechanism by the engine avoids SW from re-programming the state across context switches.

Each engine has its own hardware state variables and hence they have different context state formats. A context run on a Render engine can't be submitted to Blitter engine and vice-versa and holds true for any other engines.

Context Submission:

A context is submitted to an engine for execution by writing the context descriptor to the Execlist Submit Port (ELSP). Refer ELSP for more details. Context descriptor provides the Context ID, Address space, Logical Context Address and context valid. Refer context descriptor for more details.

Logical context address points to the context state in global virtual memory which has ring buffer details, address space setup details and other important hardware state initialization for the corresponding context. Refer Logical Context Format for more details.

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.

Context Descriptor Format

Context Descriptor Format

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

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

Programming Note on Context ID field in the Context Descriptor

This section describes the current usage by SW.

General Layout:

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

Eng. ID = Engine ID (a software defined enum to identify RCS, BCS etc.)

SW Counter = Submission Counter. (SW generates a unique counter value on every submission to ensure GroupID + PASID is unique to avoid ambiguity in fault reporting & handling)

Bit 20 = Is Proxy submission. If Set to true, SW Context ID[19:0] = LRCA [31:20], else it is an index into the Context Pool.

Direct Submission

Every application gets one context ID of their own.

SW Context ID + Engine ID + SW Counter forms the unique number

The Engine ID is used to identify which engine of a given context needs to be put into wait or ready state based on Semphore/Page Fault ID value in Semaphore/Page fault FIFO.

This method allows the context to submit work to other engines while its blocked on one.

Proxy Submission

KMD creates one context for submitting work on behalf of various user mode contexts (user mode application is not using direct submission model).

This method has certain key restrictions and behaviors:

- Work (LRCA) submitted will be scheduled on the CS in the order it was received.
- KMD uses its SW Context ID in [63:32] but uses the LRCA of the user mode context.
    - KMD's LRCA is not used for any work submission.
- If a workload hits a wait event, it does not lose its position in the schedule queue.
    - Enforces "in order" ness.
- Due to in order execution, same engine - different context semaphore synchronization is not possible.
    - Therefore, cross engine sync is simple because it clears the semaphore of the head.
- Due to in order execution, page fault on a context cannot allow a different context on same engine to execute (may preempt to idle as a power optimization).

This method allows a clean SW architecture to have KMD submissions and Ring 3 submissions to co-exist.

Logical Ring Context Format

Context descriptor has the graphics virtual address pointing to the logical context in memory. Logical context has all the details required for an engine to execute a context. This is the only means through which software can pass on all the required information to hardware for executing a context. Engine on selecting a context for execution will restore (fetch-context restore) the logical context from memory to

setup the appropriate state in the hardware. Engine on switching out the context from execution saves (store- context save) the latest updated state to logical context in memory, the updated state is result of the command buffer execution.

The Logical Context of each engine (Render, Video, Blitter, Video Enhancement, etc.) primarily consists of the following sections:

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

Per-Process of HW status Page (PPHWSP)

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

Logical Ring Context

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

Engine Context

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

- When a context is submitted for the first time for execution, SW can inhibit engine from restoring engine context by setting the "Engine Context Restore Inhibit" bit in CTXT_SR_CTL register of the logical ring context. This will avoid software from populating the Engine Context. Software must program all the state required to initialize the engine in the ring buffer which would initialize the hardware state. On a subsequent context save engine will populate the engine context with appropriate values.

- Above method can be used to create a complete logical context with engine context populated by the hardware. This Logical context can be used as a Golden Context Image or template for subsequently created contexts.

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

The detailed format of the logical ring context for Blitter, Video, and VideoEnhancement is documented in the Memory Object Overview/Logical Contexts chapter.

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

RINGBUF -- Ring Buffer Registers

Register
RING_BUFFER_TAIL - Ring Buffer Tail
RING_BUFFER_HEAD - Ring Buffer Head
RING_BUFFER_START - Ring Buffer Start
RING_BUFFER_CTL - Ring Buffer Control

Command Stream Virtual Memory Control

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

Context Status

Hardware reports the change in state of context execution to software (scheduler) through Context Status Dword. Soft-Ware can read the context status dword from time to time to track the state of context execution in hardware. A context switch reason (Context Switch Status) quad-word (64bits) is reported to the Soft-Ware (scheduler) on a valid context getting switched out. Context switch could be a synchronous context switch (from one valid element to the other valid element in the EQ) or asynchronous context switch (Load-switching from the current executing context to the very first valid element of the newly updated EQ or on Preempt to Idle). Context switch reason is also reported on HW executing the very first valid element from EQ coming out of idle indicating hardware has gone busy from idle state (Idle to Active). Context ID reported in Context Status Dword on Idle-to-Active context switch is undefined and note that there aren't any active contexts running in hardware coming out of reset, power-on or idle.

A context switch reason reported is always followed by generation of a context switch interrupt to notify the Soft-Ware about the context switch. Soft-Ware can selectively mask the context switch status being reported and the corresponding interrupt due to a specific context switch reason. Refer Context Status Report controls section for more details.

- 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. 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.

- **Format of Context Status QWord**
- **Context Status**
- Context Status should be inferred as described in the table below.
- **IDLE_CTXID Encoding**

IDLE_CTXID
0x7FF

S.No	Switch to New Queue	Ctid Away	Ctxid To	Switch Detail	Description
1	1	IDLE_CTXID	0xAB	0	**Idle to Active** Ctxid Away: IDLE_CTXID indicates HW was idle when switched to the new queue. Ctxid To: 0xAB is the context picked form the newly submitted queue to execute.
2	1	0xAB	IDLE_CTXID	0-5	**Preempt to Idle** Ctxid Away: 0xAB is the context that got switched out due to Preempt To Idle. Ctxid To: IDLE_CTXID indicates HW will go Idle following this context switch. Switch To New Queue field status set distinguishes between Preempt To Idle Vs Active To Idle Switch.
3	1	IDLE_CTXID	IDLE_CTXID	X	**Preempt To Idle, Idle to Active** Preempt To Idle has occurred when HW was idle. Ctxid Away: IDLE_CTXID indicates HW was idle when switched to the new queue (Preempt To Idle). Ctxid To: IDLE_CTXID indicates HW will go Idle following this context switch.
4	1	0xAB	0x7BC	0	**Switched to New queue and also the earlier context is complete.** Ctxid Away: 0xAB is the context that got switched out due to submission of new queue and also the context is complete. Ctxid To: 0x7BC is the context picked form the newly submitted queue to execute.
5	1	0xAB	0x7BC	5	**Switched to new queue with the earlier context preempted.** Ctxid Away: 0xAB is the context that got preempted and switched

					out due to submission of new queue. Ctxid To: 0x7BC is the context picked form the newly submitted queue to execute.
6	1	0xABC	0x7BC	1-4	**Switched to new queue, at the time of switch, executing context was waiting on an un-successful wait.** Ctxid Away: 0xAB is the context that got switched out on an un-successful wait due to submission of new queue. Ctxid To: 0x7BC is the context picked form the newly submitted queue to execute.
7	1	0xAB	0xAB	-NA-	**Lite restore.** Switched to new queue.
8	0	0xAB	0x7AC	0	**Element (Synchronous context) switch on context complete.** Ctxid Away: 0xAB is the context that got switched out due to context complete. Ctxid To: 0x7AC is the next element (context) form the execution queue selected to execute.
9	0	0xAB	0x7AC	1-4	**Element (Synchronous context) switch on un-successful wait.** Ctxid Away: 0xAC is the context that got switched out due to un-successful wait. Ctxid To: 0x7AC is the next element (context) form the execution queue selected to execute.
10	0	0xAB	IDLE_CTXID	0-4, 5*	**Active to Idle.** Ctxid Away: 0xAB is the context that got switched out due to context complete or un-successful wait. Ctxid To: IDLE_CTXID indicates HW will go Idle following this context switch. Switch To New Queue field reset status distinguishes between Active To Idle Switch Vs Preempt To Idle. Switch Detail as 5 is possible on Preempt to Idle.

Context Status Buffer in Global Hardware Status Page

Status QWords are written to the Context Status Buffer in Global Hardware Status Page at incrementing locations starting from DWORD offset of 28h. 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. The status QWs can be examined to determine the contexts executed by the hardware and the reason for switching out. The most recent location updated in the Context Status Buffer is indicated by the **Last Written Status Offset** in Global Hardware Status page at DWORD offset 47h.

Refer Global Hardware Status Page Layout at **Hardware Status Page Layout**.

Number of Context Status Entries

Number of Status Entries
12 (QW) Entries

Format of the Context Status Buffer starting at DWORD offset 28h in Global Hardware Status page

QW	Description
15	**Last Written Status Offset.** The lower byte of this QWord is written on every context switch with the (pre-increment) value of the **Context Status Buffer Write Pointer**. The lower 4 bits increment for every status Qword write; bits[7:4] are reserved and must be '0'. The lowest 4 bits indicate which of the Context Status Qwords was just written. The rest of the bits [63:8] are reserved.
14:12	Reserved: MBZ.
11:0	**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 QW11 has been written, the pointer wraps around so that the next status is written at QW0. Format = ContextStatusDW

Controls for Context Switch Status Reporting

This section describes various configuration bits available which control the hardware reporting mechanism of Context Switch Status.

Hardware reports context switch reason through context switch status report mechanism on every context switch. "Context Status Buffer Interrupt Mask" register provides mechanism to selectively mask/un-mask the context switch interrupt and the context switch status report for a given context switch reason. Hardware will not generate a context switch interrupt and context switch status report on a context switch reason that is masked in "Context Status Buffer Interrupt Mask" register. Every context switch reason reported by hardware may not be of interest to the scheduler. Scheduler may selectively mas/un-mask the context switch reasons of its interest to get notified.

Context Status Buffer Interrupt Mask Register

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.

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 table lists Preemptable Commands in ExecList mode of scheduling:

Command Streamer Preemptable Commands

Preemptable Command	Condition
MI_ARB_CHECK	AP
Element Boundary	AP (if allowed)
Semaphore Wait	Unsuccessful & AP
Wait for Event	Unsuccessful & AP (if allowed)

Table Notes:

AP - Allow Preemption if arbitration is enabled.

For additional preemptable commands specific to any engine type, refer to the engine specific command interface documentation.

Execution Status

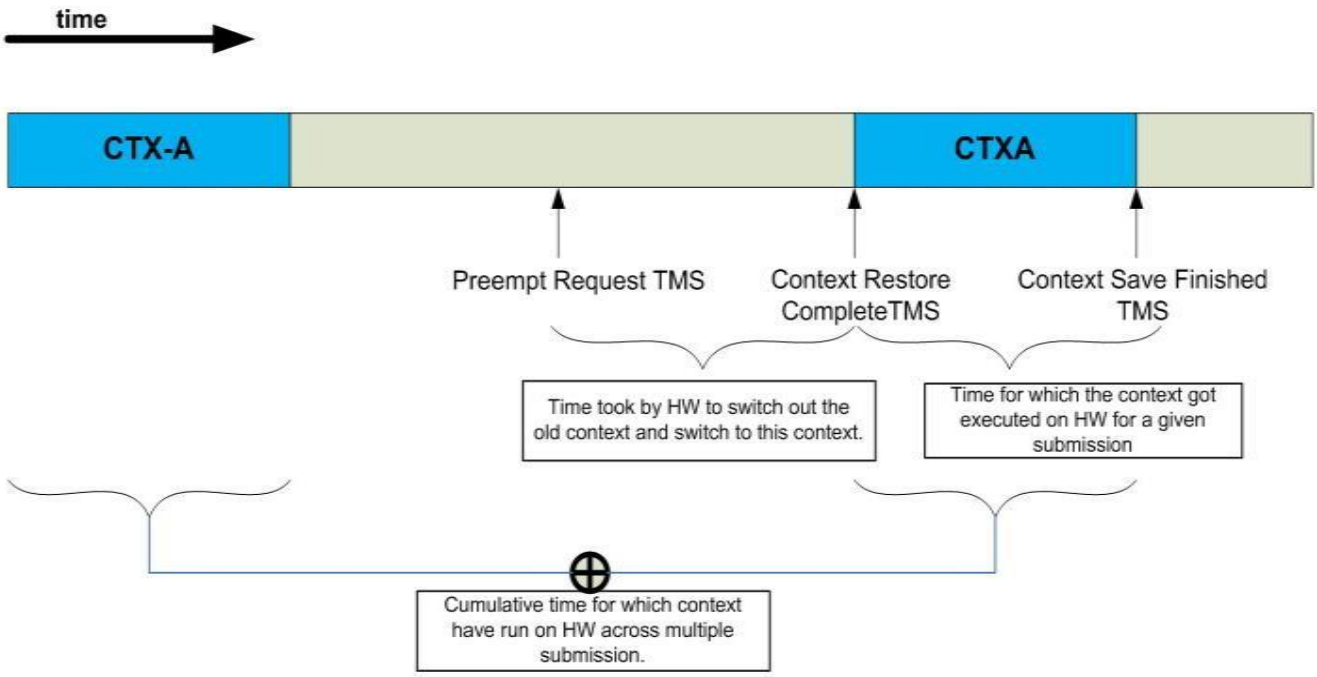
This section describes the infrastructure used to report status that the hardware provides

The Per-Process Hardware Status Page

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

The DWord offset values in the PPHWSP_LAYOUT are in decimal.

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



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

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

Hardware Status Page

The hardware status page is a naturally aligned 4KB page residing in memory. This page exists primarily to allow the device to report status via GGTT writes.

The address of this page is programmed via the HWS_PGA MI register. The definition of that register (in *Memory Interface Registers*) includes a description of the layout of the Hardware Status Page.

Interrupt Control Registers

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

Bit Defintion for Interrupt Control Registers:

Engine Interrupt Vector Definition Table

Registers
Blitter Interrupt Vector
Render Engine Interrupt Vector
VideoDecoder Interrupt Vector
VideoEnhancement Interrupt Vector
Compute Engine Interrupt Vector

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

Command Streamer > Hardware Status Mask Register

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

This section defines the Hardware-Detected Error bit definitions and ordering that is common to the EIR, EMR, and ESR registers. The EMR selects which error conditions (bits) in the ESR are reported in the EIR. Any bit set in the EIR will cause the 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:

The following structures describe the Hardware-Detected Error bits:

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
ComputeCS Hardware-Detected Error Bit Definitions Structure

The following are the EIR, EMR and ESR registers:

Registes
EIR - Error Identity Register
EMR - Error Mask Register
ESR - Error Status Register

Commands and Programming Interface

This section describes the command supported by command streamer and the programming interface.

Command Buffers

Instructions to be executed by an engine are submitted to the hardware using command buffers.

Command Ring Buffers

Command ring buffers are the memory areas used to pass instructions to the device. Refer to the Programming Interface chapter for a description of how these buffers are used to transport instructions.

The RINGBUF register sets (defined in Memory Interface Registers) are used to specify the ring buffer memory areas. The ring buffer must start on a 4KB boundary and be allocated in linear memory. The length of any one ring buffer is limited to 2MB.

Command Batch Buffers

Command batch buffers are contiguous streams of instructions referenced via an MI_BATCH_BUFFER_START and related instructions (see Memory Interface Instructions, Programming Interface). They are used to transport instructions external to ring buffers.

Programming Note	
Context:	Command batch buffers in memory objects
Batch buffers can be tagged with any memory type when produced by IA. If WB memory type is used, it should be tagged with "snoop required" for GPU consumption (to trigger snoop from CPU cache).	

Programming Note	
Context:	Command batch buffers in memory objects
The batch buffer must be QWord aligned and a multiple of QWords in length. The ending address is the address of the last valid QWord in the buffer. The length of any single batch buffer is "virtually unlimited" (i.e., could theoretically be 4GB in length).	

Workaround Batch Buffers

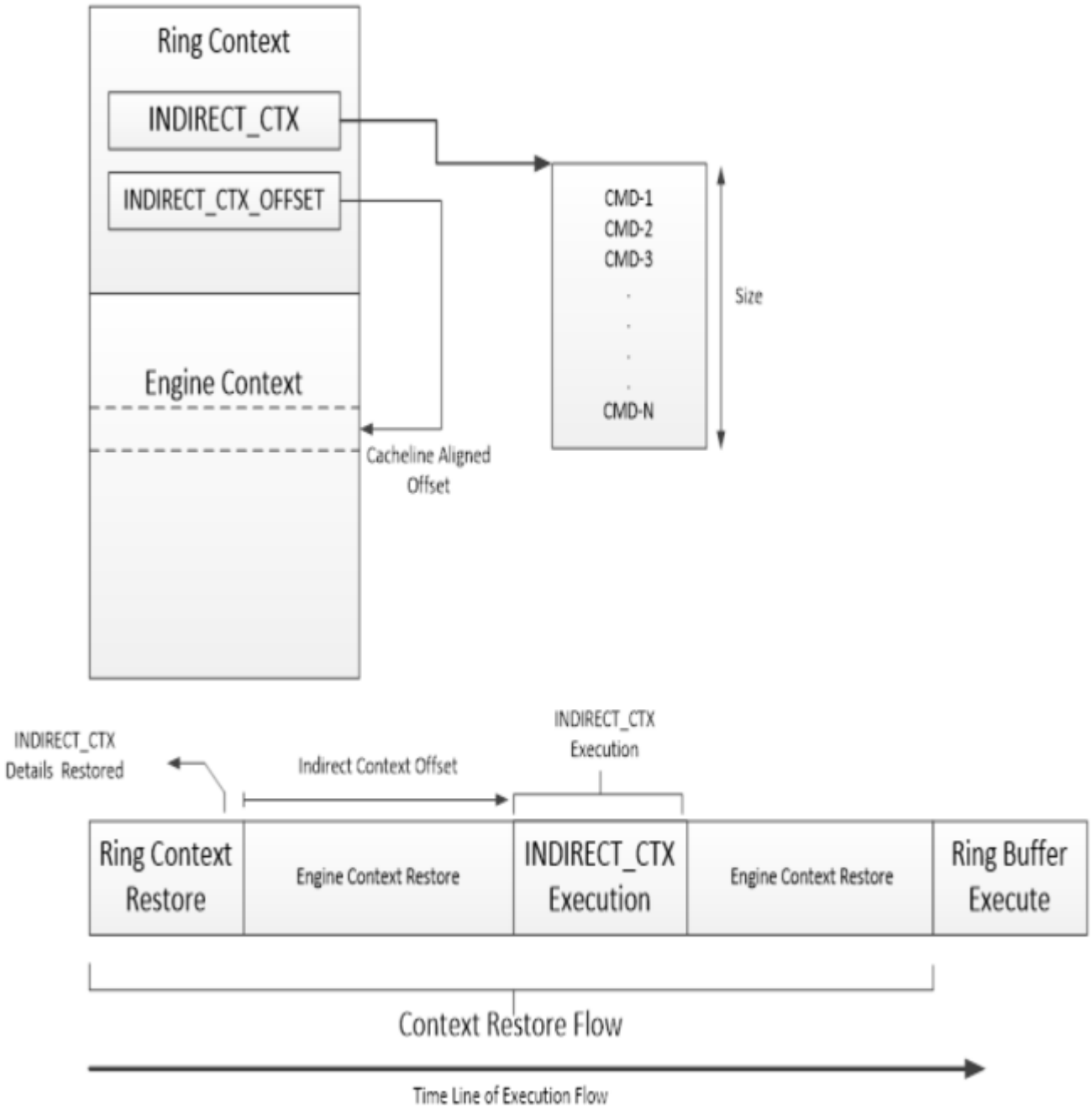
A Workaround batch buffer is a set of commands that is run by the hardware during context load time. i.e. when Command Streamer hardware is restoring the state of the context that it is about to execute

(before execution of any command in the ring buffer). The Workaround batch buffer uses pointers to command buffers that are setup by the Kernel Mode driver in the context image.

Two flavors of Workaround batch buffers are supported by the hardware. They differ in terms of exactly when the supplied workaround commands are executed in the context restore process. The mechanisms supported are:

Indirect Context Pointer (INDIRECT_CTX)

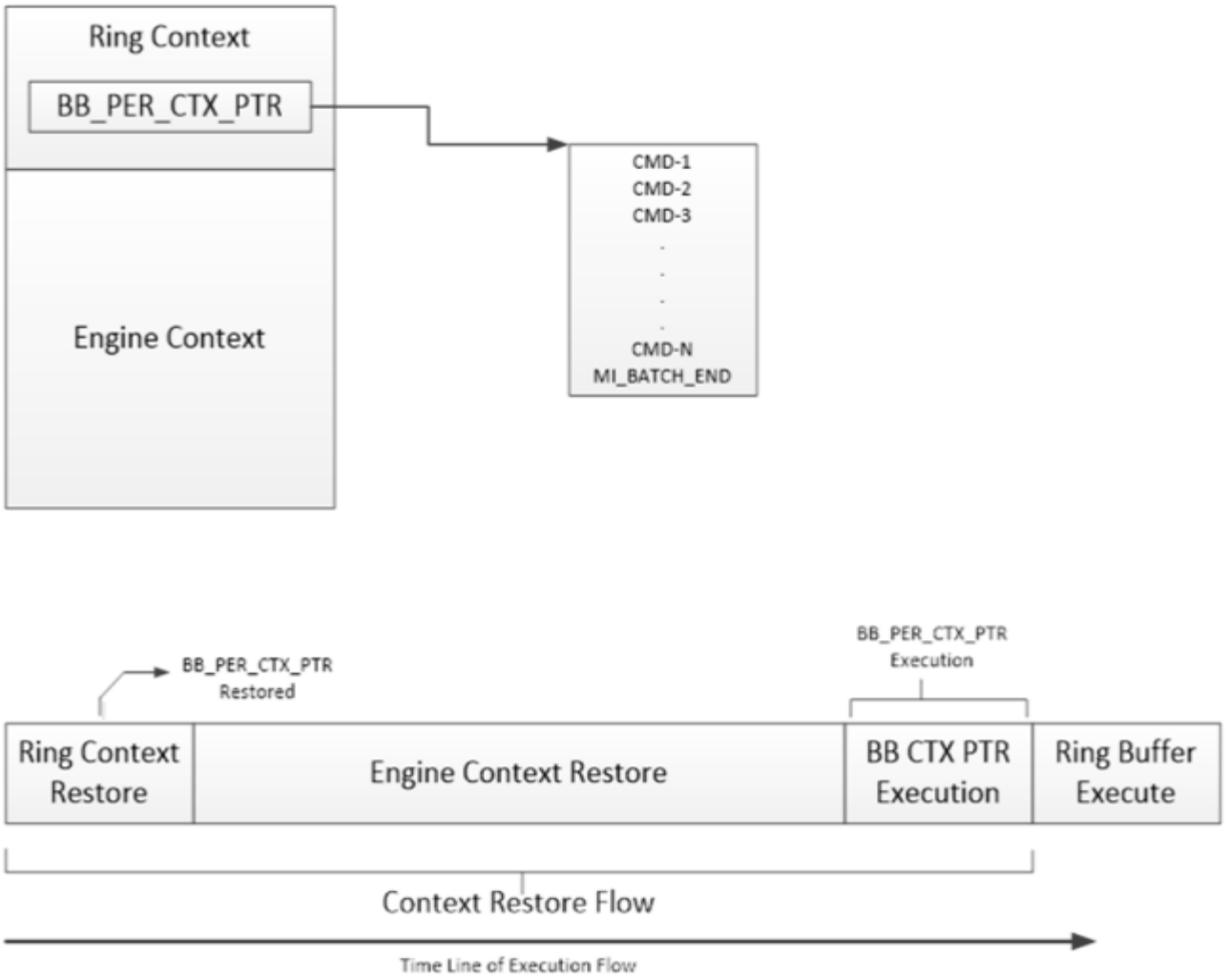
As shown in the figure below, this workaround buffer can be invoked at any cacheline aligned offset in the engine context.



Command streamer, when enabled through "INDIRECT_CTX" provides a mechanism to pause executing context restore on a given cacheline aligned offset in the engine context image and execute a command sequence from a command buffer before resuming context restore flow. This command buffer execution during context restore is referred to as "Indirect Context Pointer" execution. The start address and the size of the command buffer to be executed is provided through "INDIRECT_CTX" register and the offset in the engine context restore is provided through "INDIRECT_CTX_OFFSET". "INDIRECT_CTX" and "INDIRECT_CTX_OFFSET" registers are part of the context image and gets restored as part of the given context's context restore flow, these registers are part of the ring context image which are prior to engine context restore and hence the requirement of the offset being in engine context restore. "Indirect Context Pointer" is always in the GGTT address space of the virtual function or physical function from which the context is submitted. "Indirect context pointer" can be programmed differently for each context providing flexibility to execute different command sequence as part of "Indirect Context Pointer" execution during context restore flows.

Post Context Restore Workaround Batch Buffer

As shown in the figure, this workaround buffer is invoked at the end of the context restore.



Command streamer, when enabled through "BB_PER_CTX_PTR" provides a mechanism to execute a command sequence from a batch buffer at the end of the context restore flow during context switch process. This batch buffer is referred to as "Context Restore Batch Buffer". The batch start address for the "Context Restore Batch Buffer" gets programmed through "BB_PER_CTX_PTR", which is part of the context image and gets restored as part of the given context's context restore flow. "Context Restore Batch Buffer" execution begins (like a regular batch buffer) after the completion of fetching and execution of all the commands for the context restore flow. "Context Restore Batch Buffer" execution ends on executing MI_BATCH_BUFFER_END in the command sequence. "Context Restore Batch Buffer" is always in the GGTT address space of the virtual function or physical function from which the context is submitted. "BB_PER_CTX_PTR" can be programmed differently for every context giving flexibility to execute different command sequence (batch buffers) as part of "Context Restore Batch Buffer" execution or can be programmed to disable execution of the "Context Restore Batch Buffer" for a given context.

This mechanism is especially helpful in programming a set of commands/state that has to be always executed prior to executing a workload from a context every time it is submitted to HW for execution. Limited capability is built for "Context Restore Batch Buffer" unlike a regular MI_BATCH_BUFFER_START due to envisioned usage model, refer BB_PER_CTX_PTR for detailed programming notes.

Command Streamer Command Formats

This section describes the general format of the command streamer commands.

Command streamer 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.

Command streamer 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 command streamer 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

Engine Command Header Format

Type	Bits	
	31:29	28:0
Memory Interface (MI)	000	
Engine Command	010, 011	
Reserved	001, 100, 101, 110, 111	

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	All
02h	MI_USER_INTERRUPT	All
03h	MI_WAIT_FOR_EVENT	Render, Blitter
04h	MI_WAIT_FOR_EVENT_2	Render, Blitter
05h	MI_ARB_CHECK	All
07h	MI_REPORT_HEAD	All
08h	MI_ARB_ON_OFF	All except Blitter
0Ah	MI_BATCH_BUFFER_END	All
0Bh	MI_SUSPEND_FLUSH	All
0Ch	MI_PREDICATE	Render
2+ DWord		
10h	Reserved	
12h	MI_LOAD_SCAN_LINES_INCL	Render and Blitter
13h	MI_LOAD_SCAN_LINES_EXCL	Render and Blitter

Opcode (28:23)	Command	Pipes
14h	MI_DISPLAY_FLIP	Render and Blitter
15h	Reserved	
17h	Reserved	
18h	MI_SET_CONTEXT	Render
1Ah	MI_MATH	All
1Bh	MI_SEMAPHORE_SIGNAL	All
1Ch	MI_SEMAPHORE_WAIT	All
1Dh	MI_FORCE_WAKEUP	All except Render
1Fh	Reserved	
Store Data		
20h	MI_STORE_DATA_IMM	All
21h	MI_STORE_DATA_INDEX	All
22h	MI_LOAD_REGISTER_IMM	All
23h	MI_UPDATE_GTT	All
24h	MI_STORE_REGISTER_MEM	All
26h	MI_FLUSH_DW	All except Render
27h	MI_CLFLUSH	Render
29h	MI_LOAD_REGISTER_MEM	All
2Ah	MI_LOAD_REGISTER_REG	All
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	Reserved	All
39h-3Fh	Reserved	

Execution Control Infrastructure

This section describes the hardware infrastructure that can be used to control command execution.

Watchdog Timers

Watchdog Counter Control

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

This register is context saved as part of engine context.

Watchdog Counter Threshold

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

This register is context saved as part of engine context.

Watchdog Counter

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

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

- Watchdog timer is enabled.
- Context is valid

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

This register is not context saved and restored.

Predication

Predicate Render Registers

Register
MI_PREDICATE_SRC0 - Predicate Rendering Temporary Register0
MI_PREDICATE_SRC1 - Predicate Rendering Temporary Register1
MI_PREDICATE_DATA - Predicate Rendering Data Storage
MI_PREDICATE_RESULT - Predicate Rendering Data Result
MI_PREDICATE_RESULT_1 - Predicate Rendering Data Result 1
MI_PREDICATE_RESULT_2 - Predicate Rendering Data Result 2

MI_SET_PREDICATE

MI_SET_PREDICATE is a command that allows the driver to conditionally execute or skip a command during execution time, as detailed in the instruction definition:

The following is a list of commands that can be programmed when the PREDICATE ENABLE field in MI_SET_PREDICATE allows predication. Commands not listed here will have undefined behavior when executed with predication enabled:

Command
3DSTATE_URB_VS
3DSTATE_URB_HS
3DSTATE_URB_DS
3DSTATE_URB_GS
3DSTATE_PUSH_CONSTANT_ALLOC_VS
3DSTATE_PUSH_CONSTANT_ALLOC_HS
3DSTATE_PUSH_CONSTANT_ALLOC_DS
3DSTATE_PUSH_CONSTANT_ALLOC_GS
3DSTATE_PUSH_CONSTANT_ALLOC_PS
MI_LOAD_REGISTER_IMM
MI_STORE_DATA_IMM
3DSTATE_WM_HZ_OP
MEDIA_VFE_STATE
MEDIA_OBJECT
MEDIA_OBJECT_WALKER
MEDIA_INTERFACE_DESCRIPTOR_LOAD

MI_PREDICATE

The MI_PREDICATE command is used to control the Predicate state bit, which in turn can be used to enable/disable the processing of 3DPRIMITIVE commands.

MI_PREDICATE

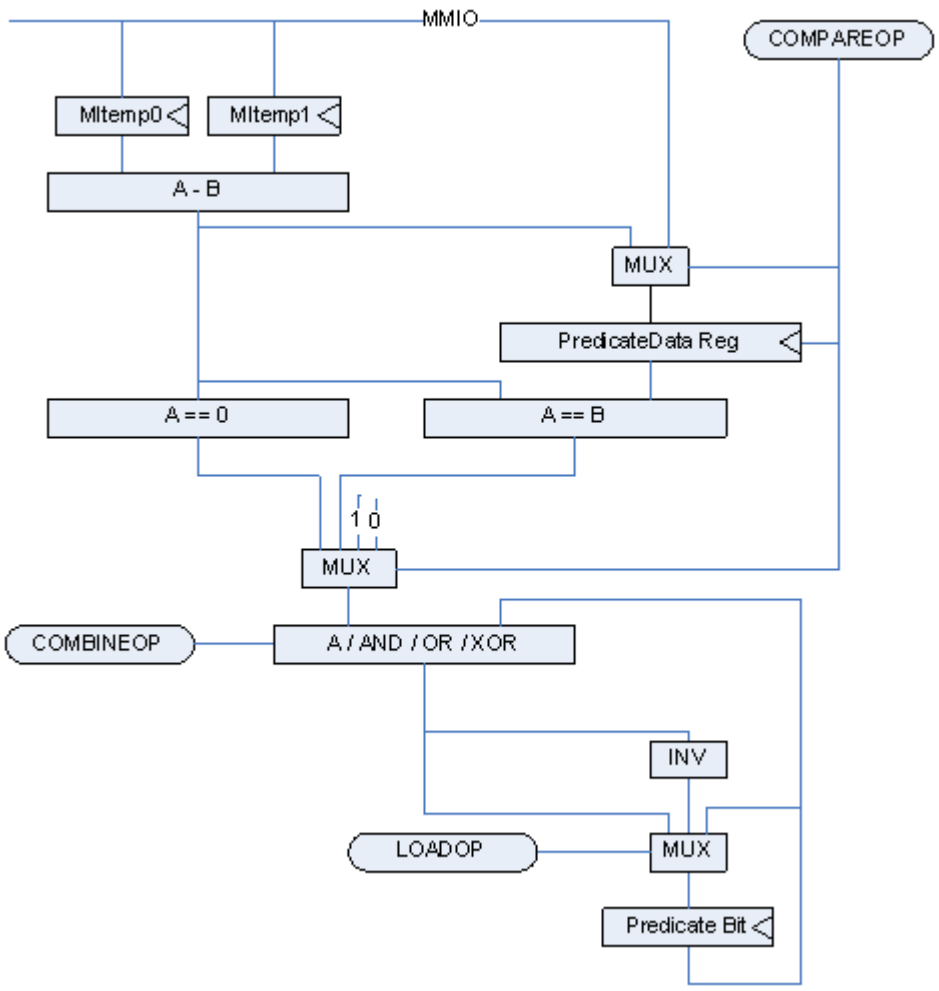
Predicated Rendering Support in HW

DX10 defines predicated rendering, where sequences of rendering commands can be discarded based on the result of a previous predicate test. A new state bit, Predicate, has been added to the command stream. In addition, a PredicateEnable bit is added to 3DPRIMITIVE. When the PredicateEnable bit is set, the command is ignored if the Predicate state bit is set.

A new command, MI_PREDICATE, is added. It contains several control fields which specify how the Predicate bit is generated.

Refer to the diagram below and the command description (linked above) for details.

MI_PREDICATE Function



MI_LOAD_REGISTER_MEM commands can be used to load the MItemp0, MItemp1, and PredicateData registers prior to MI_PREDICATE. To ensure the memory sources of the MI_LOAD_REGISTER_MEM commands are coherent with previous 3D_PIPECONTROL store-DWord operations, software can use the new **Pipe Control Flush Enable** bit in the PIPE_CONTROL command.

CS ALU Programming and Design

Command streamer implements a rudimentary ALU which supports basic Arithmetic (Addition and Subtraction) and logical operations (AND, OR, XOR) on two 64bit operands. ALU has two 64bit registers at the input SRCA and SRCB to which the operands should be loaded on which operations will be performed and outputted to a 64 bit Accumulator. Zero Flag and Carry Flag are set based on accumulator output.

Access to this ALU is thru the **MI_MATH** command.

CS_GPR - Command Streamer General Purpose Registers

Following is the Command Streamer General Purpose Register:

CS_GPR - General Purpose Register

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.

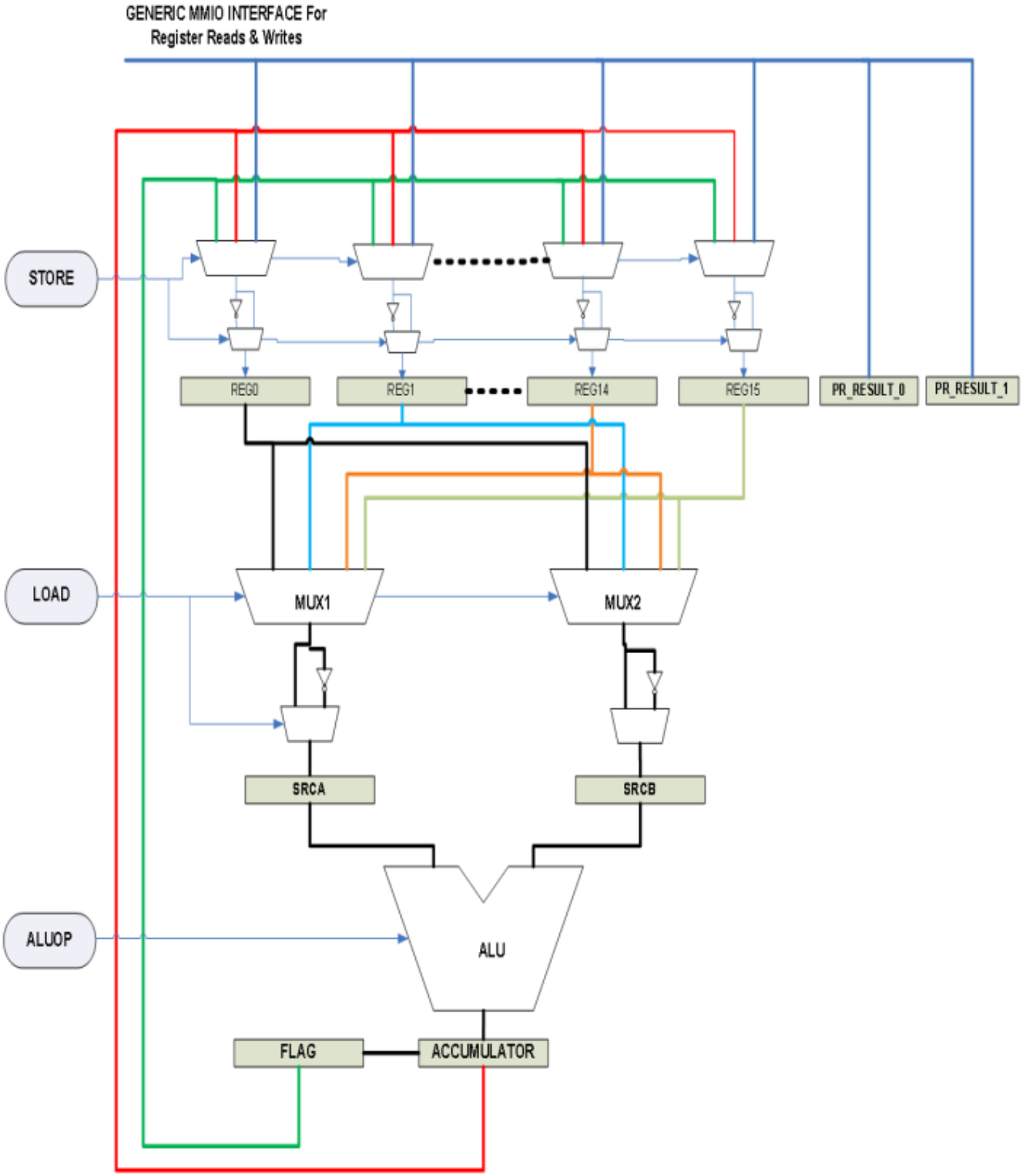
See the ALU Programming section in the Render Engine Command Streamer, for a description of the ALU programming model. Programming model is the same for all command streamers that support ALU, but each command streamer uses its own MMIO address range to address the registers. The following subsections describe the ALU registers and the programming details.

CS ALU Programming and Design

Generic Purpose Registers

Command streamer implements sixteen 64-bit General-Purpose Registers which are MMIO mapped. These registers can be accessed similar to any other MMIO mapped registers through LRI, SRM, LRR, LRM or CPU access path for reads and writes. These registers will be labeled as R0, R1, ... R15 throughout the discussion. Refer table in the B-spec update section mapping these registers to corresponding MMIO offset. A selected GPR register can be moved to SRCA or SRCB register using "LOAD" instruction. Outputs of the ALU, Accumulator, ZF and CF can be moved to any of the GPR using "STORE" instruction.

ALU BLOCK Diagram



Instruction Set

The instructions supported by the ALU can be broadly categorized into three groups:

- To move data from GPR to SRCA/SRCB - LOAD instruction.
- To move data from ACCUMULATOR/CF/ZF to GPR - STORE Instruction.
- To do arithmetic/Logical operations on SRCA and SRCB of ALU - ADD/SUB/AND/XOR/OR. Note: Accumulator is loaded with value of SRCA - SRCB on a subtraction.

Instruction Format

Each instruction is one Dword in size and consists of an ALU OPCODE, OPERAND1 and OPERAND2 in the format shown below.

ALU OPCODE	Operand-1	Operand-2
12 bits	10 bits	10 bits

NOOP and FENCE Operations

NOOP operation has does no operation but will delay and add operation idle time between commands.

Opcode	Operand1	Operand2
31:20	19:10	9:0
NOOP	N/A	N/A

Arithmetic/Logical Operations

ADD, SUB, AND, OR, and XOR are the Arithmetic and Logical operations supported by Arithmetic Logic Unit (ALU). When opcode corresponding to a logical operation is performed on SRCA and SRCB, the result is sent to ACCUMULATOR (ACCU), CF and ZF. Note that ACCU is 64-bit register. A NOOP when submitted to the ALU doesn't do anything, it is meant for creating bubble or kill cycles.

Opcode	Operand1	Operand2
31:20	19:10	9:0
ADD	N/A	N/A
SUB	N/A	N/A
OR	N/A	N/A
XOR	N/A	N/A

LOAD Operation

The LOAD instruction moves the content of the destination register (Operand2) into the source register (Operand1). The destination register can be any of the GPR (R0, R1, ..., R15) and the source registers are SRCA and SRCB of the ALU. This is the only means SRCA and SRCB can be programmed.

LOAD has different flavors, wherein one can load the inverted version of the source register into the destination register or a hard-coded value of all Zeros and All ones.

```
 // Loads any of Reg0 to Reg15 into the SRCA or SRCB registers of ALU.

     LOAD <SRCA, SRCB>, <REG0..REG15>

 // Loads inverted (bit wise) value of the mentioned Reg0 to 15 into SRCA or SRCB registers of
ALU.

     LOADINV <SRCA, SRCB>, <REG0..REG15>

 // Loads "0" into SRCA or SRCB

     LOAD0 <SRCA, SRCB>

 // Loads "1" into SRCA or SRCB

     LOAD1 <SRCA, SRCB>
```

Opcode	Operand1	Operand2
31:20	19:10	9:0
LOAD	SRCA/SRCB	R0,R1..R15
LOADINV	SRCA/SRCB	R0,R1..R15
LOAD0	SRCA/SRCB	N/A
LOAD1	SRCA/SRCB	N/A

STORE Operation

The STORE instruction moves the content of the destination register (Operand2) into the source register (Operand1). The source register can be accumulator (ACCU), CF or ZF. STORE has different flavors, wherein one can load the inverted version of the source register into destination register via STOREINV. When CF or ZF are stored, the same value is replicated on all 64 bits.

```
 // Loads ACCMULATOR or Carry Flag or Zero Flag in to any of the generic registers
 // Reg0 to Reg16. In case of CF and ZF same value is replicated on all the 64 bits.

     STORE   <R0.. R15>, <ACCU, CF, ZF >

 // Loads inverted (ACCMULATOR or Carry Flag or Zero Flag) in to any of the
 // generic registers Reg0 to Reg15.

     STOREINV <R0.. R15>, <ACCU, CF, ZF>
```

Opcode	Operand1	Operand2
31:20	19:10	9:0
STORE	R0,R1..R15	ACCU/ZF/CF
STOREINV	R0,R1..R15	ACCU/ZF/CF

Summary for ALU

Total Opcodes Supported: 12

Total Addressable Registers as source or destination: 21

- 16 GPR (R0, R1 ...R15)
- 1 ACCU
- 1ZF
- 1CF
- SRCA, SRCB

Summary of Instructions Supported

31 20	19 10	9 0
Opcode	**Operand1**	**Operand2**
LOAD	SRCA/SRCB	REG0..REG15
LOADINV	SRCA/SRCB	REG0..REG15
LOAD0	SRCA/SRCB	N/A
LOAD1	SRCA/SRCB	N?A
ADD	N/A	N/A
SUB	N/A	N/A
AND	N/A	N/A
OR	N/A	N/A
XOR	N/A	N/A
NOOP	N/A	N/A
STORE	REG0..REG15	ACCU/CF/ZF
STOREINV	REG0..REG15	ACCU/CF/ZF

Table for ALU OPCODE Encodings

In the above-mentioned table, ALU Opcode Encodings look like random numbers. The rationale behind those encodings is because the ALU Opcode is further broken down into sub-sections for ease-of-design implementation.

PREFIX		**OPCODE**		**SUBOPCODE**	
11	10	9	7	6	0
PREFIX VALUE		**Description**			
0		Regular			
1		Invert			
OPCODE VALUE		**Description**			
0		NOOP			
1		LOAD			

PREFIX	OPCODE		SUBOPCODE	
2	ALU			
3	STORE			

ALU OPCODE	OPCODE ENCODING	PREFIX(11:10)	OPCODE(9:7)	SUB-OPCODE(6:0)
NOOP	0x000	0	0	0
LOAD	0x080	0	1	0
LOADINV	0x480	1	1	0
LOAD0	0x081	0	1	1
LOAD1	0x481	1	1	1
ADD	0x100	0	2	0
SUB	0x101	0	2	1
AND	0x102	0	2	2
OR	0x103	0	2	3
XOR	0x104	0	2	4
STORE	0x180	0	3	0
STOREINV	0x580	1	3	0

Table for Register Encodings

Register	Register Encoding
R0	0x0
R1	0x1
R2	0x2
R3	0x3
R4	0x4
R5	0x5
R6	0x6
R7	0x7
R8	0x8
R9	0x9
R10	0xa
R11	0xb
R12	0xc
R13	0xd
R14	0xe
R15	0xf
SRCA	0x20
SRCB	0x21
ACCU	0x31

Register	Register Encoding
ZF	0x32
CF	0x33

MI Commands for Graphics Processing Engines

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

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

Register Access and User Mode Privileges

This section describes access to the MMIO internal to the GPU and funny I/O and how to access the ranges. Command streamer limits accesses for commands that are executed out of a PPGTT batch buffer. This is also referred to a non-privilege command buffer.

Below are the Base Addresses of each command streamer and engine blocks. While this is not all the ranges, it is the ones used to reference which registers are accessible or restricted by command streamer.

Unit	MMIO Base Offset	Description
RCS	0x2000	Render Command Streamer
POCS	0x18000	Position Command Streamer
BCS	0x22000	Blitter Command Streamer
CCS0	0x1A000	Compute Command Streamer 0
VCS/MFC	0x1C0000	Video Command Streamer 0
VCS1/MFC	0x1C4000	Video Command Streamer 1
VCS2/MFC	0x1D0000	Video Command Streamer 2
VCS3/MFC	0x1D4000	Video Command Streamer 3
VCS4/MFC	0x1E0000	Video Command Streamer 4
VCS5/MFC	0x1E4000	Video Command Streamer 5
VCS6/MFC	0x1F0000	Video Command Streamer 6
VCS7/MFC	0x1F4000	Video Command Streamer 7
VECS/MFC	0x1C8000	Video Enhancement Command Streamer 0
VECS1	0x1D8000	Video Enhancement Command Streamer 1
VECS2	0x1E8000	Video Enhancement Command Streamer 2
VECS3	0x1F8000	Video Enhancement Command Streamer 3
AV1/VDBOX0	0x1C2B00	AV1/Video Decode Block
AV1/VDBOX1	0x1C6B00	
AV1/VDBOX2	0x1D2B00	
AV1/VDBOX3	0x1D6B00	
AV1/VDBOX4	0x1E2B00	
AV1/VDBOX5	0x1E6B00	
AV1/VDBOX6	0x1F2B00	
AV1/VDBOX7	0x1F6B00	
HEVC	0x1C2800	
HEVC1	0x1C6800	
HEVC2	0x1D2800	
HEVC3	0x1D6800	
HEVC4	0x1E2800	
HEVC5	0x1E6800	

Unit	MMIO Base Offset	Description
HEVC6	0x1F2800	
HEVC7	0x1F6800	
VDENC	0x1c2d00	
VDENC1	0x1c6d00	
VDENC2	0x1d2d00	
VDENC3	0x1d6d00	
VDENC4	0x1e2d00	
VDENC5	0x1e6d00	
VDENC6	0x1f2d00	
VDENC7	0x1f6d00	

Read Only User Mode Privilege MMIO Access

The tables below specify the offsets that are allowed for MMIO reads within a non-privileged batch buffer (PPGTT). This is in addition to what is already allow listed for writes in the User Mode Privileged Commands section. Refer to Register Access and User Mode Privileges section for Base address for the below offsets.

CS means all command streamers.

Read Only Whitelist

Name	Base Address (default=none)	MMIO Offset (hex)	Size in DW
All Command Streamers			
OAG_PERF_<x>		2700	64
OAG_PERF_<x>		2B00	320
OAG_PERF_<x>		D900	192
RenderCS / PositionCS			
GPU_TIMESTAMP		2358	2
GPU_TIMESTAMP		18358	2
CS_ENGINE_ID		1808C	1
RP_STATUS0		A01C	1
OAR_PERF_<x>		2800	192
GFXREG_GT		145040	7
GFXREG_IA		145828	13
GFXREG_IO		145928	24
GFXREG_UNSLICE_FF_CTRL_FLC_THRSHLD1		A288	1
GFXREG_UNSLICE_FF_CTRL_FLC_THRSHLD2		A28C	1
GFXREG_UNSLICE_FF_COUNT1		A538	1
GFXREG_UNSLICE_FF_COUNT2		A53C	1
GFXREG_RPPREVUP		A058	1

Name	Base Address (default=none)	MMIO Offset (hex)	Size in DW
GFXREG_RPPREVDN		A064	1
GFXREG_RPUPEI		A068	1
GFXREG_RPDNEI		A06C	1
GFXREG_GT_GFX_RC6		138108	1
GFXREG_GT_GFX_RC6P		13810C	1
CS_CTX_TIMESTAMP		23A8	1
OASTATUS		DAFC	1
OAHEADPTR		DB00	1
OATAILPTR		DB04	1
ComputeCS			
GPU_TIMESTAMP	CCS	358	2
RP_STATUS0		A01C	1
GFXREG_GT		145040	7
GFXREG_IA		145828	13
GFXREG_IO		145928	24
GFXREG_UNSLICE_FF_CTRL_FLC_THRSHLD1		A288	1
GFXREG_UNSLICE_FF_CTRL_FLC_THRSHLD2		A28C	1
GFXREG_UNSLICE_FF_COUNT1		A538	1
GFXREG_UNSLICE_FF_COUNT2		A53C	1
GFXREG_RPPREVUP		A058	1
GFXREG_RPPREVDN		A064	1
GFXREG_RPUPEI		A068	1
GFXREG_RPDNEI		A06C	1
GFXREG_GT_GFX_RC6		138108	1
GFXREG_GT_GFX_RC6P		13810C	1
CS_CTX_TIMESTAMP	CCS	3A8	1
OASTATUS		DAFC	1
OAHEADPTR		DB00	1
OATAILPTR		DB04	1
OAC_PERF_<x>		15000	160
BlitterCS			
GPU_TIMESTAMP		22358	2
RP_STATUS0		A01C	1
PERFCNT1_LSB		91B8	1
PERFCNT1_MSB		91BC	1
PERFCNT2_LSB		91C0	1
PERFCNT2_MSB		91C4	1
GFXREG_GT		145040	7

Name	Base Address (default=none)	MMIO Offset (hex)	Size in DW
GFXREG_IA		145828	13
GFXREG_IO		145928	24
GFXREG_UNSLICE_FF_CTRL_FLC_THRSHLD1		A288	1
GFXREG_UNSLICE_FF_CTRL_FLC_THRSHLD2		A28C	1
GFXREG_UNSLICE_FF_COUNT1		A538	1
GFXREG_UNSLICE_FF_COUNT2		A53C	1
GFXREG_RPPREVUP		A058	1
GFXREG_RPPREVDN		A064	1
GFXREG_RPUPEI		A068	1
GFXREG_RPDNEI		A06C	1
GFXREG_GT_GFX_RC6		138108	1
GFXREG_GT_GFX_RC6P		13810C	1
OASTATUS		DAFC	1
OAHEADPTR		DB00	1
OATAILPTR		DB04	1
VideoCS			
GPU_TIMESTAMP	VCS	358	2
RP_STATUS0		A01C	1
PERFCNT1_LSB		91B8	1
PERFCNT1_MSB		91BC	1
PERFCNT2_LSB		91C0	1
PERFCNT2_MSB		91C4	1
GFXREG_GT		145040	7
GFXREG_IA		145828	13
GFXREG_IO		145928	24
GFXREG_UNSLICE_FF_CTRL_FLC_THRSHLD1		A288	1
GFXREG_UNSLICE_FF_CTRL_FLC_THRSHLD2		A28C	1
GFXREG_UNSLICE_FF_COUNT1		A538	1
GFXREG_UNSLICE_FF_COUNT2		A53C	1
GFXREG_RPPREVUP		A058	1
GFXREG_RPPREVDN		A064	1
GFXREG_RPUPEI		A068	1
GFXREG_RPDNEI		A06C	1
GFXREG_GT_GFX_RC6		138108	1
GFXREG_GT_GFX_RC6P		13810C	1
hucStatusRegOffset	HUC	0	1
hucUKernelHdrInfoRegOffset	HUC	14	1
hucStatus2RegOffset	HUC	3B0	1

Name	Base Address (default=none)	MMIO Offset (hex)	Size in DW
CS_ENGINE_ID	VCS	8C	1
OASTATUS		DAFC	1
OAHEADPTR		DB00	1
OATAILPTR		DB04	1
Perf Profiler Timer Reg		D00	1
VideoEnhancementCS			
GPU_TIMESTAMP	VECS	358	2
RP_STATUS0		A01C	1
PERFCNT1_LSB		91B8	1
PERFCNT1_MSB		91BC	1
PERFCNT2_LSB		91C0	1
PERFCNT2_MSB		91C4	1
GFXREG_GT		145040	7
GFXREG_IA		145828	13
GFXREG_IO		145928	24
GFXREG_UNSLICE_FF_CTRL_FLC_THRSHLD1		A288	1
GFXREG_UNSLICE_FF_CTRL_FLC_THRSHLD2		A28C	1
GFXREG_UNSLICE_FF_COUNT1		A538	1
GFXREG_UNSLICE_FF_COUNT2		A53C	1
GFXREG_RPPREVUP		A058	1
GFXREG_RPPREVDN		A064	1
GFXREG_RPUPEI		A068	1
GFXREG_RPDNEI		A06C	1
GFXREG_GT_GFX_RC6		138108	1
GFXREG_GT_GFX_RC6P		13810C	1
CS_ENGINE_ID	VECS	8C	1
OASTATUS		DAFC	1
OAHEADPTR		DB00	1
OATAILPTR		DB04	1

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 Violation Error" interrupt to SW. Privilege access violation checks in HW can be

disabled by setting "Privilege Check Disable" bit in GFX_MODE register. When privilege access checks are disabled HW executes the Privilege command as expected.

User Mode Privileged Commands

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

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

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

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

The tables below also are part of the allowed registers allowed to be read by a non-Privileged(PPGTT) batch buffer. Refer to Read Only User Mode Privilege MMIO Access section for the rest of the allowable registers for read access.

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

MMIO Name	MMIO Offset	Size in DWords
Cache_Mode_0	0x7000	1
Cache_Mode_1	0x7004	1
GT_MODE	0x7008	1
NOPID	0x2094	1
NOPID (POCS)	0x18094	1
INSTPM	0x20C0	1
INSTPM (POCS)	0x180C0	1
IA_VERTICES_COUNT	0x2310	2
IA_VERTICES_COUNT (POSH)	0x18310	2
IA_PRIMIVTIVES_COUNT	0x2318	2
IA_PRIMITIVES_COUNT (POSH)	0x18318	2
VS_INVOCATION_COUNT	0x2320	2
VS_INVOCATION_COUNT (POSH)	0x18320	2
HS_INVOCATION_COUNT	0x2300	2
DS_INVOCATION_COUNT	0x2308	2
GS_INVOCATION_COUNT	0x2328	2
GS_PRIMITIVES_COUNT	0x2330	2
SO_NUM_PRIMS_WRITTEN0	0x5200	2
SO_NUM_PRIMS_WRITTEN1	0x5208	2
SO_NUM_PRIMS_WRITTEN2	0x5210	2
SO_NUM_PRIMS_WRITTEN3	0x5218	2
SO_PRIM_STORAGE_NEEDED0	0x5240	2

MMIO Name	MMIO Offset	Size in DWords
SO_PRIM_STORAGE_NEEDED1	0x5248	2
SO_PRIM_STORAGE_NEEDED2	0x5250	2
SO_PRIM_STORAGE_NEEDED3	0x5258	2
SO_WRITE_OFFSET0	0x5280	1
SO_WRITE_OFFSET1	0x5284	1
SO_WRITE_OFFSET2	0x5288	1
SO_WRITE_OFFSET3	0x528C	1
CL_INVOCATION_COUNT	0x2338	2
CL_INVOCATION_COUNT (POSH)	0x18338	2
CL_PRIMITIVES_COUNT	0x2340	2
CL_PRIMITIVES_COUNT (POSH)	0x18340	2
PS_INVOCATION_COUNT	0x2348	2
PS_DEPTH_COUNT	0x2350	2
PS_INVOCATION_COUNT_0	0x22C8	2
PS_DEPTH_COUNT _0	0x22D8	2
PS_INVOCATION_COUNT_1	0x22F0	2
PS_DEPTH_COUNT _1	0x22F8	2
PS_INVOCATION_COUNT_2	0x2448	2
PS_DEPTH_COUNT_2	0x2450	2
PS_INVOCATION_COUNT_3	0x2458	2
PS_DEPTH_COUNT_3	0x2460	2
PS_INVOCATION_COUNT_4	0x2468	2
PS_DEPTH_COUNT_4	0x2470	2
PS_INVOCATION_COUNT_5	0x24A0	2
PS_DEPTH_COUNT_5	0x24A8	2
PS_INVOCATION_COUNT_6	0x25D0	2
PS_DEPTH_COUNT_6	0x25B0	2
PS_INVOCATION_COUNT_7	0x25D8	2
PS_DEPTH_COUNT_7	0x25B8	2
CPS_INVOCATION_COUNT	0x2478	2
GPUGPU_DISPATCHDIMX	0x2500	1
GPUGPU_DISPATCHDIMY	0x2504	1
GPUGPU_DISPATCHDIMZ	0x2508	1
MI_PREDICATE_SRC0	0x2400	1
MI_PREDICATE_SRC0 (POSH)	0x18400	1
MI_PREDICATE_SRC0	0x2404	1
MI_PREDICATE_SRC0 (POSH)	0x18404	
MI_PREDICATE_SRC1	0x2408	1

MMIO Name	MMIO Offset	Size in DWords
MI_PREDICATE_SRC1 (POSH)	0x18408	
MI_PREDICATE_SRC1	0x240C	1
MI_PREDICATE_SRC1 (POSH)	0x1840C	
MI_PREDICATE_DATA	0x2410	1
MI_PREDICATE_DATA (POSH)	0x18410	
MI_PREDICATE_DATA	0x2414	1
MI_PREDICATE_DATA (POSH)	0x18414	
MI_PREDICATE_RESULT	0x2418	1
MI_PREDICATE_RESULT (POSH)	0x18418	
MI_PREDICATE_RESULT_1	0x241C	1
MI_PREDICATE_RESULT_1 (POSH)	0x1841C	
MI_PREDICATE_RESULT_2	0x23BC	1
MI_PREDICATE_RESULT_2 (POSH)	0x183BC	
3DPRIM_END_OFFSET	0x2420	1
3DPRIM_END_OFFSET (POSH)	0x18420	1
3DPRIM_START_VERTEX	0x2430	1
3DPRIM_START_VERTEX (POSH)	0x18430	1
3DPRIM_VERTEX_COUNT	0x2434	1
3DPRIM_VERTEX_COUNT (POSH)	0x18434	1
3DPRIM_INSTANCE_COUNT	0x2438	1
3DPRIM_INSTANCE_COUNT (POSH)	0x18438	1
3DPRIM_START_INSTANCE	0x243C	1
3DPRIM_START_INSTANCE (POSH)	0x1843C	1
3DPRIM_BASE_VERTEX	0x2440	1
3DPRIM_BASE_VERTEX (POSH)	0x18440	1
3DPRIM_XP0	0x2690	1
3DPRIM_XP0 (POSH)	0x18690	1
3DPRIM_XP1	0x2694	1
3DPRIM_XP1 (POSH)	0x18694	1
3DPRIM_XP2	0x2698	1
3DPRIM_XP2 (POSH)	0x18698	1
GPGPU_THREADS_DISPATCHED	0x2290	2
BB_OFFSET	0x2158	1
BB_OFFSET (POCS)	0x18158	1
CS_GPR (1-16)	0x2600	32
CS_GPR (1-16) (POSH)	0x18600	32
OA_CTX_CONTROL	0x2360	1
OA_CTX_CONTROL_MSG	0x2AA0	1

MMIO Name	MMIO Offset	Size in DWords
OACTXID	0x2364	1
OAR_OACONTROL	0x2960	1
OAR_OASTATUS	0x2968	1
PR_CTR_CTL_RCSUNIT	0x2178	1
PR_CTR_THRSH_RCSUNIT	0x217C	1
VSR_PUSH_CONSTANT_BASE	0xE518	1
PTBR_PAGE_POOL_SIZE_REGISTER	0x18590	1
PSS_MODE	0x7038	1
CMD_BUFF_CTL	0x2084	1
Z_DISCARD_EN	0x7040	1
AUX_TABLE_BASE_ADDR_LOW	0x4200	1
AUX_TABLE_BASE_ADDR_HIGH	0x4204	1
CCS_AUX_INV	0x4208	1
TRTT_CR	0x4400	1
TRTT_VA_RANGE	0x4404	1
TRTT_L3_BASE_LOW	0x4408	1
TRTT_L3_BASE_HIGH	0x440C	1
TR_NULL_GFX	0x4410	1
TRTT_INVAL	0x4414	1
LSQCREG1	0xB100	1
LSQCREG4	0xB118	1
LSQCREG5	0xB158	1
LSQCREG6	0xB15C	1
L3ALLOCREG	0xB134	1
L3TCCNTLREG	0xB138	1

User Mode Non-Privileged Registers for Compute Command Streamer (CCS)

MMIO Name	MMIO Offset	Size in DWords
NOPID	0x1A094	1
INSTPM	0x1A0C0	1
GPUGPU_DISPATCHDIMX	0x1A500	1
GPUGPU_DISPATCHDIMY	0x1A504	1
GPUGPU_DISPATCHDIMZ	0x1A508	1
MI_PREDICATE_SRC0	0x1A400	1
MI_PREDICATE_SRC0	0x1A404	1
MI_PREDICATE_SRC1	0x1A408	1
MI_PREDICATE_SRC1	0x1A40C	1
MI_PREDICATE_DATA	0x1A410	1
MI_PREDICATE_DATA	0x1A414	1
MI_PREDICATE_RESULT	0x1A418	1
MI_PREDICATE_RESULT_1	0x1A41C	1
MI_PREDICATE_RESULT_2	0x1A3BC	1
GPGPU_THREADS_DISPATCHED	0x1A290	2
BB_OFFSET	0x1A158	1
CS_GPR (1-16)	0x1A600	32
PR_CTR_CTL_RCSUNIT	0x1A178	1
PR_CTR_THRSH_RCSUNIT	0x1A17C	1
CMD_BUFF_CTL	0x1A084	1
COMPCS0_AUX_TABLE_BASE_ADDR_LOW	0x42C0	1
COMPCS0_AUX_TABLE_BASE_ADDR_HIGH	0x42C4	1
COMPCS0_CCS_AUX_NV	0x42C8	1
COMP_CTX0_TRTT_CR	0x4580	1
COMP_CTX0_TRTT_VA_RANGE	0x4584	1
COMP_CTX0_TRTT_L3_BASE_LOW	0x4588	1
COMP_CTX0_TRTT_L3_BASE_HIGH	0x458C	1
COMP_CTX0_TRTT_NULL	0x4590	1
COMP_CTX0_TRTT_INVAL	0x4594	1

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

MMIO Name	MMIO Offset	Size in DWords
BCS_GPR	0x22600	32
BCS_SWCTRL	0x22200	1
BLIT_CCTL	0x22204	1
PR_CTR_CTL_BCSUNIT	0x22178	1
PR_CTR_THRSH_BCSUNIT	0x2217C	1
BLT_TRTT_CR	0x4480	1
BLT_TRTT_VA_RANGE	0x4484	1
BLT_TRTT_L3_BASE_LOW	0x4488	1
BLT_TRTT_L3_BASE_HIGH	0x448C	1
BLT_TRTT_NULL	0x4490	1
BLT_TRTT_INV	0x4494	1
NOPID	0x22094	1
MI_PREDICATE_RESULT_1	0x2241C	1
MI_PREDICATE_RESULT_2	0x223BC	1
INSTPM	0x220C0	1

Refer to Register Access and User Mode Privileges section for Base address for the below offsets.

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

MMIO Name	MMIO Base	MMIO Offset	Size in DWords
VECS_GPR	VECS	0x600	32
PR_CTR_CTL_VECSUNIT	VECS	0x178	1
PR_CTR_THRSH_VECSUNIT	VECS	0x17C	1
NOPID	VECS	0x094	1
MI_PREDICATE_RESULT_1	VECS	0x41C	1
MI_PREDICATE_RESULT_2	VECS	0x3BC	1
INSTPM	VECS	0x0C0	1

* These registers are not at a standard offset from their corresponding CS MMIO base address and hence are stated individually per CS in a separate table below.

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

MMIO Name	Unit Base	MMIO Range	Size in DWords
VCS_GPR	VCS	0x600	32
PR_CTR_CTL_VCSUNIT	VCS	0x178	1
PR_CTR_THRSH_VCSUNIT	VCS	0x17C	1
MFC_VDBOX1	VCS	0x800	512
HEVC	HEVC	0x00	64
VDENC	VDENC	0x00	64
NOPID	VCS	0x094	1
MI_PREDICATE_RESULT_1	VCS	0x41C	1
MI_PREDICATE_RESULT_2	VCS	0x3BC	1
INSTPM	VCS	0x0C0	1

* These registers are not at a standard offset from their corresponding CS MMIO base address and hence are stated individually per CS in a separate table below.

VEBOX-1

MMIO Name	MMIO Base	MMIO Offset	Size in DWords
AUX_TABLE_BASE_ADDR_LOW*	n/a	0x42B0	1
AUX_TABLE_BASE_ADDR_HIGH*	n/a	0x42B4	1
CCS_AUX_INV*	n/a	0x42B8	1
TRTT_CR*	n/a	0x4560	1
TRTT_VA_RANGE*	n/a	0x4564	1
TRTT_L3_BASE_LOW*	n/a	0x4568	1
TRTT_L3_BASE_HIGH*	n/a	0x456C	1
TRTT_NULL*	n/a	0x4570	1
TRTT_INVAL*	n/a	0x4474	1

VDBOX-2

MMIO Name	MMIO Base	MMIO Offset	Size in DWords
AUX_TABLE_BASE_ADDR_LOW*	n/a	0x4290	1
AUX_TABLE_BASE_ADDR_HIGH*	n/a	0x4294	1
CCS_AUX_INV*	n/a	0x4298	1
TRTT_CR*	n/a	0x4520	1
TRTT_VA_RANGE*	n/a	0x4524	1
TRTT_L3_BASE_LOW*	n/a	0x4528	1
TRTT_L3_BASE_HIGH*	n/a	0x452C	1
TRTT_NULL*	n/a	0x4530	1
TRTT_INVAL*	n/a	0x4534	1

VDBOX-3

MMIO Name	MMIO Base	MMIO Offset	Size in DWords
AUX_TABLE_BASE_ADDR_LOW*	n/a	0x42A0	1
AUX_TABLE_BASE_ADDR_HIGH*	n/a	0x42A4	1
CCS_AUX_INV*	n/a	0x42A8	1
TRTT_CR*	n/a	0x4540	1
TRTT_VA_RANGE*	n/a	0x4544	1
TRTT_L3_BASE_LOW*	n/a	0x4548	1
TRTT_L3_BASE_HIGH*	n/a	0x454C	1
TRTT_NULL*	n/a	0x4550	1
TRTT_INVAL*	n/a	0x4554	1

Context Management

When the scheduler submits a list of workloads through the execution list, the Command streamer hardware executes one context at a time.

An engine starts executing a context by loading the state (LRCA) in memory that is pointed to by the context descriptor.

The structure of the LRCA are described in subsequent sections.

Global State

There is only one copy of state variables across contexts running on an engine and changing the settings of these variables requires explicit programming of these state variables. Typically, global state variables are programmed only once either at the time of power-on or at the time of GFX driver initialization. Examples of global sate include:

- MI registers (HWSTAM, SEMA_WAIT_POLL ..etc)
- Configuration Registers (GFX_MODE ..etc)

The global state of an engine is context save/restored during power-off/on regimes.

Following subsections describe the power context images of engines across generations.

Power Context Image

This section lists the power context image of Video Engine, Copy Engine and Video Enhancement Engine across generations.

CSFE Power context without Display

CSFE Power Context Image with Display

Description	Offset	Unit	# of DW	Address Offset (PWR)	CSFE/CSBE
NOOP		CS	1	0000	CSFE
Load_Register_Immediate header	0x1100_00B5	CS	1	0001	CSFE
GFX_MODE	0x029C	CS	2	0002	CSFE
GHWSP	0x0080	CS	2	0004	CSFE
RC_PSMI_CONTROL	0x0050	CS	2	0006	CSFE
RC_PWRCTX_MAXCNT	0x0054	CS	2	0008	CSFE
CTX_WA_PTR	0x0058	CS	2	000A	CSFE
NOPID	0x0094	CS	2	000C	CSFE
HWSTAM	0x0098	CS	2	000E	CSFE
IMR	0x00A8	CS	2	0010	CSFE
EIR	0x00B0	CS	2	0012	CSFE
EMR	0x00B4	CS	2	0014	CSFE
CMD_CCTL_0	0x00C4	CS	2	0016	CSFE
PREEMPT_DLY	0x0214	CS	2	0018	CSFE
CTXT_PREMP_DBG	0x0248	CS	2	001A	CSFE
WAIT_FOR_RC6_EXIT	0x00CC	CS	2	001C	CSFE
RCS_CTXID_PREEMPTION_HINT	0x04CC	CS	2	001E	CSFE
CS_PREEMPTION_HINT_UDW	0x04C8	CS	2	0020	CSFE
CS_PREEMPTION_HINT	0x04BC	CS	2	0022	CSFE
CCID Register	0x0180	CS	2	0024	CSFE

Description	Offset	Unit	# of DW	Address Offset (PWR)	CSFE/CSBE
MI_PREDICATE_RESULT_2	0x03BC	CS	2	0026	CSFE
CTXT_ST_PTR	0x03A0	CS	2	0028	CSFE
CTXT_ST_BUF	0x0370	CS	24	002A	CSFE
CTXT_ST_BUF	0x03C0	CS	24	0042	CSFE
SEMA_WAIT_POLL	0x024C	CS	2	005A	CSFE
IDLEDELAY	0x023C	CS	2	005C	CSFE
RCS_FORCE_TO_NONPRIV_0_11	0x04D0	CS	24	005E	CSFE
RCS_FORCE_TO_NONPRIV_12_15	0x010	CS	8	0076	CSFE
RCS_FORCE_TO_NONPRIV_16_19	0x1D0	CS	8	007E	CSFE
EXECLIST_STATUS_REGISTER	0x0234	CS	2	0086	CSFE
CXT_OFFSET	0x01AC	CS	2	008A	CSBE
STOP_PARSER_CONTROL	0x0424	CS	2	008C	CSBE
STOP_PARSER_HINT_ADDR	0x0428	Cs	4	008E	CSBE
EXECLIST_SQ_CONTENTS	0x0510-0x054F	CS	32	0092	CSFE
CSB_INTERRUPT_MASK	0x0218	CS	2	00B2	CSFE
EQ_ELEMENT_MASK	0x056C	CS	2	00B4	CSFE
NOOP		CS	8	00B8	CSFE
NOOP		CS	2	00B8	CSFE

CSFE Power Context with Display

CSFE Power Context Image with Display

Description	Offset	Unit	# of DW	Address Offset (PWR)	CSFE/CSBE
NOOP		CS	1	0000	CSFE
Load_Register_Immediate header	0x1100_00C7	CS	1	0001	CSFE
GFX_MODE	0x029C	CS	2	0002	CSFE
GHWSP	0x0080	CS	2	0004	CSFE
RC_PSMI_CONTROL	0x0050	CS	2	0006	CSFE
RC_PWRCTX_MAXCNT	0x0054	CS	2	0008	CSFE
CTX_WA_PTR	0x0058	CS	2	000A	CSFE
NOPID	0x0094	CS	2	000C	CSFE
HWSTAM	0x0098	CS	2	000E	CSFE
IMR	0x00A8	CS	2	0010	CSFE
EIR	0x00B0	CS	2	0012	CSFE
EMR	0x00B4	CS	2	0014	CSFE
CMD_CCTL_0	0x00C4	CS	2	0016	CSFE
PREEMPT_DLY	0x0214	CS	2	0018	CSFE
CTXT_PREMP_DBG	0x0248	CS	2	001A	CSFE

Description	Offset	Unit	# of DW	Address Offset (PWR)	CSFE/CSBE
SYNC_FLIP_STATUS	0x02D0	CS	2	001C	CSFE
SYNC_FLIP_STATUS_1	0x02D4	CS	2	001E	CSFE
SYNC_FLIP_STATUS_2	0x02EC	CS	2	0020	CSFE
WAIT_FOR_RC6_EXIT	0x00CC	CS	2	0022	CSFE
RCS_CTXID_PREEMPTION_HINT	0x04CC	CS	2	0024	CSFE
CS_PREEMPTION_HINT_UDW	0x04C8	CS	2	0026	CSFE
CS_PREEMPTION_HINT	0x04BC	CS	2	0028	CSFE
CCID Register	0x0180	CS	2	002A	CSFE
MI_PREDICATE_RESULT_2	0x03BC	CS	2	002C	CSFE
CTXT_ST_PTR	0x03A0	CS	2	002E	CSFE
CTXT_ST_BUF	0x0370	CS	24	0030	CSFE
CTXT_ST_BUF	0x03C0	CS	24	0048	CSFE
SEMA_WAIT_POLL	0x024C	CS	2	0060	CSFE
IDLEDELAY	0x023C	CS	2	0062	CSFE
DISPLAY MESSAGE FORWARD STATUS	0x02E8	CS	2	0064	CSFE
RCS_FORCE_TO_NONPRIV_0_11	0x04D0	CS	24	0066	CSFE
RCS_FORCE_TO_NONPRIV_12_15	0x010	CS	8	007E	CSFE
RCS_FORCE_TO_NONPRIV_16_19	0x1D0	CS	8	0086	CSFE
EXECLIST_STATUS_REGISTER	0x0234	CS	2	008E	CSFE
CXT_OFFSET	0x01AC	CS	2	0092	CSBE
STOP_PARSER_CONTROL	0x0424	CS	2	0094	CSBE
STOP_PARSER_HINT_ADDR	0x0428	Cs	4	0098	CSBE
SYNC_FLIP_STATUS_3	0x02B8	CS	2	009A	CSFE
SYNC_FLIP_STATUS_4	0x02C0	CS	2	009C	CSFE
SYNC_FLIP_STATUS_5	0x02C4	CS	2	009E	CSFE
SYNC_FLIP_STATUS_6	0x01F8	CS	2	00A0	CSFE
DISPLAY MESSAGE FORWARD STATUS_2	0x0188	CS	2	00A2	CSFE
DISPLAY MESSAGE FORWARD STATUS_3	0x018C	CS	2	00A4	CSFE
EXECLIST_SQ_CONTENTS	0x0510-0x054F	CS	32	00A6	CSFE
CSB_INTERRUPT_MASK	0x0218	CS	2	00C6	CSFE
EQ_ELEMENT_MASK	0x056C	CS	2	00C8	CSFE
NOOP		CS	4	00D0	CSFE

CSFE Power Context

CSFE Power Context Image

Description	Offset	Unit	# of DW	Address Offset (PWR)	CSFE/CSBE
NOOP		CS	1	0	CSFE
Load_Register_Immediate header	0x1100_00DB	CS	1	001	CSFE
GFX_MODE	0x029C	CS	2	0002	CSFE
GHWSP	0x0080	CS	2	0004	CSFE
RING_BUFFER_CONTROL (Ring Always Disabled)	0x003C	CS	2	0006	CSFE
Ring Head Pointer Register	0x0034	CS	2	0008	CSFE
Ring Tail Pointer Register	0x0030	CS	2	000A	CSFE
RING_BUFFER_START	0x0038	CS	2	000C	CSFE
RING_BUFFER_CONTROL (Original status)	0x003C	CS	2	000E	CSFE
Batch Buffer Current Head Register (UDW)	0x0168	CS	2	0010	CSFE
Batch Buffer Current Head Register	0x0140	CS	2	0012	CSFE
Batch Buffer State Register	0x0110	CS	2	0014	CSFE
SECOND_BB_ADDR_UDW	0x011C	CS	2	0016	CSFE
SECOND_BB_ADDR	0x0114	CS	2	0018	CSFE
SECOND_BB_STATE	0x0118	CS	2	001A	CSFE
RC_PSMI_CONTROL	0x0050	CS	2	001C	CSFE
RC_PWRCTX_MAXCNT	0x0054	CS	2	001E	CSFE
CTX_WA_PTR	0x0058	CS	2	0020	CSFE
NOPID	0x0094	CS	2	0022	CSFE
HWSTAM	0x0098	CS	2	0024	CSFE
IMR	0x00A8	CS	2	0026	CSFE
EIR	0x00B0	CS	2	0028	CSFE
EMR	0x00B4	CS	2	002A	CSFE
CMD_CCTL_0	0x00C4	CS	2	002C	CSFE
UHPTR	0x0134	CS	2	002E	CSFE
BB_PREEMPT_ADDR_UDW	0x016C	CS	2	0030	CSFE
BB_PREEMPT_ADDR	0x0148	CS	2	0032	CSFE
RING_BUFFER_HEAD_PREEMPT_REG	0x014C	CS	2	0034	CSFE
PREEMPT_DLY	0x0214	CS	2	0036	CSFE
CTXT_PREMP_DBG	0x0248	CS	2	0038	CSFE
SYNC_FLIP_STATUS	0x02D0	CS	2	003A	CSFE
SYNC_FLIP_STATUS_1	0x02D4	CS	2	003C	CSFE

Description	Offset	Unit	# of DW	Address Offset (PWR)	CSFE/CSBE
SYNC_FLIP_STATUS_2	0x02EC	CS	2	003E	CSFE
WAIT_FOR_RC6_EXIT	0x00CC	CS	2	0040	CSFE
RCS_CTXID_PREEMPTION_HINT	0x04CC	CS	2	0042	CSFE
CS_PREEMPTION_HINT_UDW	0x04C8	CS	2	0044	CSFE
CS_PREEMPTION_HINT	0x04BC	CS	2	0046	CSFE
CCID Register	0x0180	CS	2	0048	CSFE
SBB_PREEMPT_ADDRESS_UDW	0x0138	CS	2	004A	CSFE
SBB_PREEMPT_ADDRESS	0x013C	CS	2	004C	CSFE
MI_PREDICATE_RESULT_2	0x03BC	CS	2	004E	CSFE
CTXT_ST_PTR	0x03A0	CS	2	0050	CSFE
CTXT_ST_BUF	0x0370	CS	24	0052	CSFE
CTXT_ST_BUF	0x03C0	CS	24	006A	CSFE
SEMA_WAIT_POLL	0x024C	CS	2	0082	CSFE
IDLEDELAY	0x023C	CS	2	0084	CSFE
DISPLAY MESSAGE FORWARD STATUS	0x02E8	CS	2	0086	CSFE
RCS_FORCE_TO_NONPRIV	0x04D0	CS	24	0088	CSFE
EXECLIST_STATUS_REGISTER	0x0234	CS	2	00A0	CSFE
CXT_OFFSET	0x01AC	CS	2	00A4	CSBE
STOP_PARSER_CONTROL	0x0424	CS	2	00A6	CSBE
STOP_PARSER_HINT_ADDR	0x0428	Cs	4	00A8	CSBE
SYNC_FLIP_STATUS_3	0x02B8	CS	2	00AC	CSFE
SYNC_FLIP_STATUS_4	0x02C0	CS	2	00AE	CSFE
SYNC_FLIP_STATUS_5	0x02C4	CS	2	00B0	CSFE
SYNC_FLIP_STATUS_6	0x01F8	CS	2	00B2	CSFE
DISPLAY MESSAGE FORWARD STATUS_2	0x0188	CS	2	00B4	CSFE
DISPLAY MESSAGE FORWARD STATUS_3	0x018C	CS	2	00B6	CSFE
EXECLIST_SQ_CONTENTS	0x0510-0x054F	CS	32	00B8	CSFE
CSB_INTERRUPT_MASK	0x0218	CS	2	00D8	CSFE
EQ_ELEMENT_MASK	0x056C	CS	2	00DA	CSFE

Context State

Context state is associated with a specific context. Context state can be programmed through Command Stream only when the associated context is being actively executed in the engine. Examples of context state include

- 3D STATE * commands (Render Engine)
- BCS TILE REGISTER (Copy Engine)

Context state is save/restored through Logical Context.

Logical Contexts

A logical context is an area in memory used to store hardware context state information and the context is referenced via a context descriptor. Context descriptor carries graphics memory address. Logical Context is always in global virtual memory. GFX device provides means to save and restore the hardware context state to logical context. Context state save/restore mechanism is used by SW to avoid re-programming the HW state across context switches for a given context.

CSFE Execlist Context

This section details the CSFE Execlist Context which is the common layout referred to as part of the VDBOX, Copy Engine and Video Enhancment context images.

CSFE Execlist Context

Programming Note	
Context:	MMIO Offset information
MMIO offset mentioned for the registers in the below table are offset form the units "MMIO Base Offset" mentioned in the table " Base Offset for Video Command Streamers and Media Engine" in the section User Mode Privileged Commands. For Example: VECS has MMIO Base Offset as "0x1C_8000". In the below table "Context Control" register has 0x00244 as offset against it, actual MMIO Offset of "Context Control" register for VECS is 0xx1C_8244. Blitter Engine MMIO base offset must be considered as 0x2_0000.	

EXECLIST CONTEXT
EXECLIST CONTEXT(PPGTT Base)
ENGINE CONTEXT

Description	MMIO Offset/Command	Unit	# of DW	Offset
NOOP		CSEL	1	**0**
Load_Register_Immediate header	0x1108_1019	CSEL	1	**0001**
Context Control	0x00244	CSEL	2	**0002**
Ring Head Pointer Register	0x00034	CSEL	2	**0004**
Ring Tail Pointer Register	0x00030	CSEL	2	**0006**
RING_BUFFER_START	0x00038	CSEL	2	**0008**
RING_BUFFER_CONTROL	0x0003C	CSEL	2	**000A**
Batch Buffer Current Head Register (UDW)	0x00168	CSEL	2	**000C**
Batch Buffer Current Head Register	0x00140	CSEL	2	**000E**
Batch Buffer State Register	0x00110	CSEL	2	**0010**
BB_PER_CTX_PTR	0x001C0	CSEL	2	**0012**
CS_INDIRECT_CTX	0x001C4	CSEL	2	**0014**
CS_INDIRECT_CTX_OFFSET	0x001C8	CSEL	2	**0016**
CCID	0x00180	CSEL	2	**0018**
SEMAPHORE_TOKEN	0x002B4	CSEL	2	**001A**
NOOP		CSEL	4	**001C**
NOOP		CSEL	1	**0020**
Load_Register_Immediate header	0x1108_1011	CSEL	1	**0021**
CTX_TIMESTAMP	0x003A8	CSEL	2	**0022**
PDP3_UDW	0x0028C	CSEL	2	**0024**
PDP3_LDW	0x00288	CSEL	2	**0026**
PDP2_UDW	0x00284	CSEL	2	**0028**
PDP2_LDW	0x00280	CSEL	2	**002A**
PDP1_UDW	0x0027C	CSEL	2	**002C**
PDP1_LDW	0x00278	CSEL	2	**002E**
PDP0_UDW	0x00274	CSEL	2	**0030**
PDP0_LDW	0x00270	CSEL	2	**0032**
NOOP		CSEL	4	**0034**
NOOP		CSEL	8	**0038**
NOOP	{EXISTS IF (VCS, VECS)}	CSEL_BE	16	**0040**
NOOP	{EXISTS IF (BCS)}	CSEL_BE	1	**0040**
Load_Register_Immediate header	0x1100_1003 {EXISTS IF (BCS)}	CSEL_BE	1	**0041**
BCS TILE REGISTER	0x02200 {EXISTS IF (BCS)}	CSEL_BE	2	**0042**

Description	MMIO Offset/Command	Unit	# of DW	Offset
BLIT_CCTL	0x02204 {EXISTS IF (BCS)}	CSEL_BE	2	**0044**
NOOP	{EXISTS IF (BCS)}	CSEL_BE	10	**0046**
NOOP		CSFE	1	**0050**
Load_Register_Immediate header	0x1100_1063	CSFE	1	**0051**
BB_STACK_WRITE_PORT	0x00588	CSFE	12	**0052**
EXCC	0x00028	CSFE	2	**005E**
MI_MODE	0x0009C	CSFE	2	**0060**
INSTPM	0x000C0	CSFE	2	**0062**
PR_CTR_CTL	0x00178	CSFE	2	**0064**
PR_CTR_THRSH	0x0017C	CSFE	2	**0066**
TIMESTAMP Register (LSB)	0x00358	CSFE	2	**0068**
BB_START_ADDR_UDW	0x00170	CSFE	2	**006A**
BB_START_ADDR	0x00150	CSFE	2	**006C**
BB_ADD_DIFF	0x00154	CSFE	2	**006E**
BB_OFFSET	0x00158	CSFE	2	**0070**
MI_PREDICATE_RESULT_1	0x0041C	CSFE	2	**0072**
CS_GPR (1-16)	0x00600	CSFE	64	**0074**
IPEHR	0x00068	CSFE	2	**00B4**
MI_FORCE_WAKEUP	{EXISTS IF (VCS, VECS)}	CSFE	2	**00B6**
NOOP	{EXISTS IF (BCS)}	CSFE	2	**00B6**
AUX_TTRTT_BASE_ADDRESS_SECTION (Separate section for each engine mentioned below)		CSFE	18	**00B8**
NOOP*		CSFE	2	**00CA**
NOOP		CSFE	4	**00CC**

Video Enhancement Engine: AUX_TRTT_BASE_ADDRESS_SECTION

Description	MMIO Offset/Command	Unit	# of DW	Offset
NOOP		CSFE	1	**00B6**
Load_Register_Immediate header	0x1102_100F	CSFE	1	**00B7**
TRTT_CR	0x4460	CSFE	2	**00BC**
TRTT_VA_RANGE	0x4464	CSFE	2	**00BE**
TRTT_L3_BASE_LOW	0x4468	CSFE	2	**00C0**
TRTT_L3_BASE_HIGH	0x446C	CSFE	2	**00C2**
TRTT_NULL	0x4470	CSFE	2	**00C4**
TRTT_INVAL	0x4474	CSFE	2	**00C6**
AUX_TABLE_BASE_ADDR_LOW	0x4230	CSFE	2	**00B8**
AUX_TABLE_BASE_ADDR_HIGH	0x4234	CSFE	2	**00BA**

Video Decode Engine:AUX_TRTT_BASE_ADDRESS_SECTION

Description	MMIO Offset/Command	Unit	# of DW	Offset
NOOp		CSFE	1	**00B6**
Load_Register_Immediate header	0x1102_100F	CSFE	1	**00B7**
TRTT_CR	0x4420	CSFE	2	**00BC**
TRTT_VA_RANGE	0x4424	CSFE	2	**00BE**
TRTT_L3_BASE_LOW	0x4428	CSFE	2	**00C0**
TRTT_L3_BASE_HIGH	0x442C	CSFE	2	**00C2**
TRTT_NULL	0x4430	CSFE	2	**00C4**
TRTT_INVAL	0x4434	CSFE	2	**00C6**
AUX_TABLE_BASE_ADDR_LOW	0x4210	CSFE	2	**00B8**
AUX_TABLE_BASE_ADDR_HIGH	0x4214	CSFE	2	**00BA**

Copy Engine: AUX_TRTT_BASE_ADDRESS_SECTION

Description	MMIO Offset/Command	Unit	# of DW	Offset
NOOP		CSFE	1	**00B6**
Load_Register_Immediate header	0x1102_100B	CSFE	1	**00B7**
TRTT_CR	0x4480	CSFE	2	**00BC**
TRTT_VA_RANGE	0x4484	CSFE	2	**00BE**
TRTT_L3_BASE_LOW	0x4488	CSFE	2	**00C0**
TRTT_L3_BASE_HIGH	0x448C	CSFE	2	**00C2**
TRTT_NULL	0x4490	CSFE	2	**00C4**
TRTT_INVAL	0x4494	CSFE	2	**00C6**
NOOP		CSFE	2	**00B8**
NOOP		CSFE	2	**00BA**

Producer-Consumer Data ordering for MI Commands

This section details the explicit data ordering enforced by HW for produce-consume of data between MI commands and explicit programming notes for data ordering not explicitly enforced by HW.

This section describes the MI commands that result in modification of data in Graphics memory or MMIO registers. These commands can be treated as producers of data for which consumers can either be SW or subsequent commands (MI or non-MI) executed by HW.

Operations (memory update or MMIO update) resulting from a command execution can be classified in to posted or non-posted.

- An operation is classified as posted if the operation initiated by the command is not guaranteed to complete (data change to be reflected) before HW moves on to the following command to execute, the posted operation is guaranteed to complete eventually. Posted operations can be forced to complete through explicit or implicit means, detailed in following section.
    - For example, a memory write is called posted if the hardware moves on to the next command after generating a memory write without waiting for the memory modification to reach a global observable point.
- An operation is classified as non-posted if the operation initiated by the command is completed before HW moves on to execute the following command.
    - For example, a memory write is called non-posted if the hardware waits for the memory write to reach a global observable point before it moves on to the next command to execute.

There are certain commands which supported both posted and non-posted operations and can be programmed by SW to select the appropriate behavior based on the usage model.

Memory Data Ordering

This section details the produce-consume data for MI commands accessing memory.

Memory Data Producer

This section describes the MI commands that modify data in graphics memory. Few commands always generate posted memory writes whereas few commands provide programmable option to generate posted Vs non-posted memory writes.

- A memory write is called posted if the hardware moves on to the next command after generating a memory write and doesn't wait for the memory modification to reach a global observable point. Since HW doesn't wait for the memory write completion it can execute the next command immediately without incurring any additional latency. Read after Write hazard is applicable in this scenario.
- A memory write is called non-posted if the hardware waits for the memory write to reach a global observable point before it moves on to the next command to execute. Since HW waits for the memory write completion before it goes on to the next command, it will incur additional latency causing a stall at top of the pipe. Read after write hazard will not happen in this scenario.

A write completion of a non-posted memory write will guarantee all the prior posted memory writes are to global observable (GO) point.

For optimal performance SW must use commands generating non-posted memory writes at the minimal. For example, a single non-posted memory write can be used just before the consume point to flush out all the prior posted memory writes to global observable point. Based on the usage model SW can use a combination of commands that generate posted memory writes and non-posted memory writes for optimal performance.

Table below lists the MI Commands that can update/modify the data in graphics memory and the associated type of memory write.

Command	Memory Write Type
MI_STORE_REGISTER_MEM	Posted
MI_COPY_MEM_MEM	Posted
MI_STORE_DATA_INDEX	Posted
MI_STORE_DATA_IMM (with Wr. Completion)	Non-Posted
MI_STORE_DATA_IMM (with out Wr. Completion)	Posted
MI_REPORT_HEAD	Posted
MI_UPDATE_GTT	Posted
MI_REPORT_PERF_COUNT	Posted
MI_ATOMIC	Posted, Non-Posted
MI_FLUSH_DW (With Post-Sync Operation)	Non-Posted
PIPE_CONTROL (non-stalling, with Post-Sync Operation)	Posted
PIPE_CONTROL (Stalling, Post-Sync Operation)	Non-Posted

Apart from the MI commands that generate Non-Posted memory writes listed in the above table, execution of following commands will also implicitly ensure all prior posted writes are to Global Observable point.

Command
PIPE_CONTROL (Stalling)
MI_FLUSH_DWORD
MI_USER_INTERRUPT
PIPE_CONTROL (with Notify Interrupt)

Memory Data - Consumer

Table below lists the MI command that read the data from graphics memory as part of the command execution. Data in memory should be coherent prior to execution of these command to achieve expected functional behavior upon execution of these commands, Graphics memory writes by the earlier executed MI commands must be GO prior to execution of these commands. However, hardware has started explicitly enforcing data ordering for few of the commands (based on the prevalent usage models) and mentioned in the table below.

Command	Coherency Requirement
MI_LOAD_REGISTER_MEM	HW implicitly ensures memory writes by the prior MI commands by the corresponding engine are coherent for this command execution.
MI_BATCH_BUFFER_START	SW must ensure the data coherency.
MI_CONDITIONAL_BATCH_BUFFER_END	SW must ensure the data coherency.
MI_ATOMIC	HW implicitly ensures memory writes by the prior MI commands by the corresponding engine are coherent for this command execution.
MI_SEMAPHORE_WAIT	HW implicitly ensures memory writes by the prior MI commands by the corresponding engine are coherent for this command execution.

SW can use any of the MI commands that generate non-posted memory writes or the commands that implicitly force prior memory writes to GO to ensure data is coherent in memory prior to execution of these commands.

MMIO Data Ordering

This section details the produce-consume data for MI commands accessing MMIO registers.

MMIO Data Producer

Table below lists the MI commands that modify data in MMIO registers and also states if the MMIO writes generated are posted Vs non-posted.

- A MMIO write is called non-posted if the hardware waits for the MMIO update to occur before it moves on to the next command to execute.
- A MMIO write is called posted if the hardware moves on to the next command after generating a MMIO write without waiting for the MMIO update to occur.

All the MI commands listed below generate non-posted MMIO writes and hence HW guarantees the MMIO modification has taken place before HW moves on the following command.

MI_LOAD_REGISTER_MEM supports both posted and non-posted behavior and can be configured through "Async Mode Enable" bit in the command header.

Command	MMIO Write Type
MI_LOAD_REGISTER_IMM	Non-Posted
PIPE_CONTROL	Non-Posted
MI_LOAD_REGISTER_MEM	Posted, Non-Posted
MI_MATH	Non-Posted
MI_LOAD_REGISTER_REG	Non-Posted

MMIO Data Consumer

All the commands that modify the MMIO are non-posted and hence any MI command consumer of MMIO data will always get the latest updated value.

Software must take care of appropriately programming the "Async Mode Enable" bit in MI_LOAD_REGISTER_MEM command based on the requirements to enforce data ordering between producer and consumer. Table below lists the MI commands that consume the MMIO data.

Command
MI_STORE_REGISTER_MEM
MI_PREDICATE
MI_LOAD_REGISTER_REG
MI_MATH
MI_SET_PREDICATE
MI_SEMAPHORE_WAIT (register poll)
MI_SEMAPHPORE_SIGNAL

Command Fetch

Command parser implements a DMA engine to fetch the command data from memory. DMA engine pre-fetches eight cacheline worth of command data into its storage and keeps it ready to be executed, it keeps fetching command data as and when space is available in the storage.

Advanced Command Prefetch

Advanced command pre-fetch is an enhancement to the existing DMA engine to addresses its limitation of not being to stream pre-fetches on instructions causing jumps (Ex: Batch Buffer Start and Batch Buffer End). Advanced command pre-fetch is enabled by default; however software is given flexibility to enable or disable advanced command pre-fetch at its convenience by following means:

- Global state through MMIO (GFX_MODE) which is part of power context. This MMIO bit must be enabled or disabled only through CPU path and must be done when there is no context active in hardware.
- Inline to the command sequence through MI_ARB_CHK, this state is maintained per context. This mechanism must be used by SW to disable pre-fetch around self-modification-code or around selective command sequences of interest.

In order for pre-fetch functionality to be enabled both global and per context state should be set to pre-fetch enable.

Self Modifying Code

Self-modifying code (SMC) in context to command parser refers to a scenario where in a command sequence executed by the command parser modifies the upcoming commands to be executed by command parser. DMA pre-fetch of command data introduces certain programming restriction on placement of the SMC in the command sequence.

- The modifying commands and the modified commands should be far apart by the number of cachelines fetched by the CS for latency hiding(See Max Command FIFO Depth below) Or
- The modifying commands and the modified command must be executed after a batch buffer start(chained or nested)

Advanced command preparser adds additional limitation to the programming of self-modifying-code. Software must explicitly disable the preparser logic before programming a batch buffer whose contents has been modified by the earlier programmed command sequence (self-modifying-code). Preparser logic must be disabled using MI_ARB_CHECK command prior to programing the MI_BATCH_BUFFER_START command and pre-fetch logic must be enabled using MI_ARB_CHECK as the first command inside the batch buffer.

Max Command FIFO Depth
8

Observability

This is the Observability section.

Observability Overview

As GFX-enabled systems and usage models have grown in complexity over time, a number of hardware features have been added to provide more insight into hardware behavior while running a commercially available operating system. This chapter documents these features with pointers to relevant sections in other chapters. Supported observability features include:

Feature
Performance counters
Internal node tracing

Note: This chapter describes the registers and instructions used to monitor GPU performance. Please review other volumes in this specification to understand the terms, functionality and details for specific Intel graphics devices.

DFD Configuration Restore

Since DFD logic does not usually add value to end user usage models and its configuration space is large (which would add latency to power management restore flows), it is typically not enabled during normal

operation for optimal power & performance. Hence, additional steps are required when DFD functionality is needed in combination with system configurations where GT logic loses power/is reset. The basic strategy per scenario is detailed below.

GT Power-up/RC6 Exit

Strategy
Replicate failure without power management
Configure the DFD restore feature

Render Engine Power-up

Configure the RCS RC6 W/A batch buffer to restore render engine DFD configuration ONLY.

Media Engine Power-up

Configure the applicable media command stream W/A batch buffer to restore media engine DFD configuration ONLY.

Resume from Partial GT Power Down

For cases where SW is aware of power well state, re-apply DFD configuration.

For cases where SW is not aware of power well state, configure the per-context W/A batch buffer to apply the DFD configuration on every context load.

Trace

This section contains the following contents:

Feature
• Performance Visibility

Performance Visibility

Motivation for Hardware-Assisted Performance Visibility

As the focus on GFX performance and programmability has increased over time, the need for hardware (HW) support to rapidly identify bottlenecks in HW and efficiently tune the work sent to same has become correspondingly important. This section describes the HW support for Performance Visibility.

Performance Event Counting

An earlier generation introduced dedicated GFX performance counters to address key issues associated with existing chipset CHAPs counters (lack of synchronization with GFX rendering work and low sampling frequency achievable when sampling via CPU MMIO read). Furthermore, reliance on SoC assets created a cross-IP dependency that was difficult to manage well. Hence, the approach since that earlier generation has been to use dedicated counters managed by the graphics device driver for graphics performance

measurement. The dedicated counter values are written to memory whenever an MI_REPORT_PERF_COUNT command is placed in the ring buffer.

While this approach eliminated much of the error associated with the previous approaches, it is still limited to sampling the counters only at the boundaries between ring commands. This inherently limited the ability of performance analysis tools to drill down into a primitive, which can contain thousands of triangles and require several hundreds of milliseconds to render. It is further worth noting that precise sampling via MI_REPORT_PERF command requires flushing the GFX pipeline before and after the work of interest. The overhead of flushing the GFX pipeline can become large if the work of interest is small, hence reducing the accuracy of the performance counter measurement. In such situations, the flush can be removed, or internally triggered reporting can be used with some resulting loss of precision in which draws/dispatches are being profiled.

Additionally, Intel design and architecture teams found that the existing silicon-based performance analysis tools provided only a general idea of where a problem may exist but were not able to pinpoint a problem. This was generally because the counter values are integrated across a very large time period, washing out the dynamic behavior of the workload.

All OA config registers are tied to GT global reset and hence are not affected by per-engine resets (e.g. render only reset).

OA Programming Guidelines

SW utilizing OA HW is expected to monitor the overflow/lost report status for the OABUFFER and respond as appropriate for the active usage model.

In order for OA counters to increment the 'Counter Stop-Resume Mechanism' bit of the OACTXCONTROL register must be set. This requires a RCS context with this bit set be loaded, and either RCS force wake be enabled or the RCS context be left active for the duration of the window this counter is needed for.

In general, OA is effectively unable to count between the power context save that happens prior to GFX entering RC6 and the power context restore that occurs on the next RC6 exit. This limitation results from the fact that the counters themselves are power context save/restored and hence the counts that (may) have accumulated in this time window are overwritten by the saved values that are read back from the power context save area. An example of the kind of information that can be missed is the GTI traffic resulting from the power context save of OA itself. The size of this performance counting blind spot is microarchitecturally minimized as much as reasonably possible but still varies from device to device.

Legacy OACS functionality is now logically split into two functions called OAG (OA Global) and OAR (OA Render). Summary of the blocks is as follows:

OAG:

- Handles OA buffer and timer/internally triggered sampling.
- Is unaffected by engine reset / power well status.
- Is inaccessible by non-privileged batch buffers but accessible by all command streamers / CPU.
- Implements free-running utilization counters.
- Is GT power context save/restored.

- Is only allowed to access global GTT memory.

OAR:

- Is expected to behave as a part of the render engine from a clocking/power well/reset perspective.
- Implements MI_REPORT_PERF command.
- Is render context save/restored, making all values reported by MI_REPORT_PERF per-context.
- Must be initialized to power-on default values as part of RCS golden context creation (please refer to RCS section describing golden context creation for full details) or implementation-specific undesirable behavior may occur.
- Doesn't support timer/internally triggered sampling.
- Can be enabled/disabled independent from OAG.

Is only intended to be accessed by RCS, access from other command streamers / CPU may have implementation-specific negative side-effects.

HW Support

This section contains various reporting counters and registers for hardware support for Performance Visibility.

Performance Counter Report Formats

Counters layout for various values of select from the register:

Counters layout for various values of the "Counter Select" from the register:

Counter Select = 000

A-Cntr 10 (low dword)	A-Cntr 9 (low dword)	A-Cntr 8 (low dword)	A-Cntr 7 (low dword)	GPU_TICKS	CTX ID	TIME_STAMP	RPT_ID
A-Cntr 18 (low dword)	A-Cntr 17 (low dword)	A-Cntr 16 (low dword)	A-Cntr 15 (low dword)	A-Cntr 14 (low dword)	A-Cntr 13 (low dword)	A-Cntr 12 (low dword)	A-Cntr 11 (low dword)

Counter Select = 010

A-Cntr 10 (low dword)	A-Cntr 9 (low dword)	A-Cntr 8 (low dword)	A-Cntr 7 (low dword)	GPU_TICKS	CTX ID	TIME_STAMP	RPT_ID
A-Cntr 18 (low dword)	A-Cntr 17 (low dword)	A-Cntr 16 (low dword)	A-Cntr 15 (low dword)	A-Cntr 14 (low dword)	A-Cntr 13 (low dword)	A-Cntr 12 (low dword)	A-Cntr 11 (low dword)
B-Cntr 7	B-Cntr 6	B-Cntr 5	B-Cntr 4	B-Cntr 3	B-Cntr 2	B-Cntr 1	B-cntr 0
C-Cntr 7	C-Cntr 6	C-Cntr 5	C-Cntr 4	C-Cntr 3	C-Cntr 2	C-Cntr 1	C-Cntr 0

Counter Select = 111

C-Cntr 3	C-Cntr 2	C-Cntr 1	C-Cntr 0	GPU_TICKS	CTX ID	TIME_STAMP	RPT_ID
B-Cntr 7	B-Cntr 6	B-Cntr 5	B-Cntr 4	B-Cntr 3	B-Cntr 2	B-Cntr 1	B-Cntr 0

OAR Report Format (Counter Select = 0b101):

A-Cntr 3 (low dword)	A-Cntr 2 (low dword)	A-Cntr 1 (low dword)	A-Cntr 0 (low dword)	GPU_TICKS	CTX ID	TIME_STAMP	RPT_ID
A-Cntr 11 (low dword)	A-Cntr 10 (low dword)	A-Cntr 9 (low dword)	A-Cntr 8 (low dword)	A-Cntr 7 (low dword)	A-Cntr 6 (low dword)	A-Cntr 5 (low dword)	A-Cntr 4 (low dword)
A-Cntr 19 (low dword)	A-Cntr 18 (low dword)	A-Cntr 17 (low dword)	A-Cntr 16 (low dword)	A-Cntr 15 (low dword)	A-Cntr 14 (low dword)	A-Cntr 13 (low dword)	A-Cntr 12 (low dword)
A-Cntr 27 (low dword)	A-Cntr 26 (low dword)	A-Cntr 25 (low dword)	A-Cntr 24 (low dword)	A-Cntr 23 (low dword)	A-Cntr 22 (low dword)	A-Cntr 21 (low dword)	A-Cntr 20 (low dword)
A-Cntr 35 (low dword)	A-Cntr 34 (low dword)	A-Cntr 33 (low dword)	A-Cntr 32 (low dword)	A-Cntr 31 (low dword)	A-Cntr 30 (low dword)	A-Cntr 29 (low dword)	A-Cntr 28 (low dword)
High bytes of A31-A28	High bytes of A27-A24	High bytes of A23-A20	High bytes of A19-A16	High bytes of A15-A12	High bytes of A11-A8	High bytes of A7-A4	High bytes of A3-A0
B-Cntr 7	B-Cntr 6	B-Cntr 5	B-Cntr 4	B-Cntr 3	B-Cntr 2	B-Cntr 1	B-Cntr 0
C-Cntr 7	C-Cntr 6	C-Cntr 5	C-Cntr 4	C-Cntr 3	C-Cntr 2	C-Cntr 1	C-Cntr 0

OAG Report Format (Counter Select = 0b101)

A-Cntr 3 (low dword)	A-Cntr 2 (low dword)	A-Cntr 1 (low dword)	A-Cntr 0 (low dword)	GPU_TICKS	CTX ID	TIME_STAMP	RPT_ID
A-Cntr 11 (low dword)	A-Cntr 10 (low dword)	A-Cntr 9 (low dword)	A-Cntr 8 (low dword)	A-Cntr 7 (low dword)	A-Cntr 6 (low dword)	A-Cntr 5 (low dword)	A-Cntr 4 (low dword)
A-Cntr 19 (low dword)	A-Cntr 18 (low dword)	A-Cntr 17 (low dword)	A-Cntr 16 (low dword)	A-Cntr 15 (low dword)	A-Cntr 14 (low dword)	A-Cntr 13 (low dword)	A-Cntr 12 (low dword)
A-Cntr 27 (low dword)	A-Cntr 26 (low dword)	A-Cntr 25 (low dword)	A-Cntr 24 (low dword)	A-Cntr 23 (low dword)	A-Cntr 22 (low dword)	A-Cntr 21 (low dword)	A-Cntr 20 (low dword)
A-Cntr 35 (low dword)	A-Cntr 34 (low dword)	A-Cntr 33 (low dword)	A-Cntr 32 (low dword)	A-Cntr 31 (low dword)	A-Cntr 30 (low dword)	A-Cntr 29 (low dword)	A-Cntr 28 (low dword)
High bytes of A31-A28	High bytes of A27-A24	High bytes of A23-A20	High bytes of A19-A16	High bytes of A15-A12	High bytes of A11-A8	High bytes of A7-A4	High bytes of A3-A0
B-Cntr 7	B-Cntr 6	B-Cntr 5	B-Cntr 4	B-Cntr 3	B-Cntr 2	B-Cntr 1	B-Cntr 0
C-Cntr 7	C-Cntr 6	C-Cntr 5	C-Cntr 4	C-Cntr 3	C-Cntr 2	C-Cntr 1	C-Cntr 0

Counter Select = 111

C-Cntr 3	C-Cntr 2	C-Cntr 1	C-Cntr 0	GPU_TICKS	CTX ID	TIME_STAMP	RPT_ID
B-Cntr 7	B-Cntr 6	B-Cntr 5	B-Cntr 4	B-Cntr 3	B-Cntr 2	B-Cntr 1	B-Cntr 0

Description of RPT_ID and other important fields of the layout:

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>GPU TICKS[31:0]</td><td>GPU_TICKS is simply a free-running count of render clocks elapsed used for normalizing other counters (e.g. EU active time), it is expected that the rate that this value advances will vary with frequency and freeze (but not lose its value) when all GT clocks are gated, GT is in RC6, and so on.</td></tr>
<tr><td>Context ID[31:0]</td><td>This field carries the Context ID of the active context in render engine.
[31:0]: Context ID in Execlist mode of scheduling.</td></tr>
<tr><td>TIME_STAMP[31:0]</td><td>This field provides an elapsed real-time value that can be used as a timestamp for GPU events over short periods of time. This field has the same format at TIMESTAMP register defined in Render Command Streamer .</td></tr>
<tr><td>RPT_ID[46:38]</td><td>Reserved (for future use)</td></tr>
<tr><td>RPT_ID[35:34]</td><td>Reserved (for future Tile IDs)</td></tr>
<tr><td>RPT_ID[31:0]</td><td>This field has several sub fields as defined below:
<table>
<tr><td>31:26</td><td>SourceID[5:0]
Encoded value to identify various sources like any CS or Shader unit from which the Report was requested.
Programming note:</td></tr>
<tr><td>25:19</td><td>Report Reason[6:0]
Report_reason[0]: When set indicates current report is due to "Timer Triggered".
Report_reason[1]: When set indicates current report is due to "Internal report trigger 1".
Report_reason[2]: When set indicates current report is due to "Internal report trigger 2".
Report_reason[3]: When set indicates current report is due to "Render context switch".
Report_reason[4]: When set indicates current report is due to "GO transition from '1' to '0' ".
Report_reason[5]: When set indicates current report is due to a change in unslice/slice ratio
Report_reason[6]: When set indicates current report is due to a MMIO Trigger
Programming note:</td></tr>
<tr><td>18</td><td>Start Trigger Event:This bit is multiplexed from "Start Trigger Event-1" or "Start Trigger Event-2" based on the "Internal Report Trigger-1" or "Internal Report Trigger-2" asserted in the Report Reason respectively. "Internal Report Trigger-1" is given priority over "Internal Report Trigger-2". By default Start Trigger Event-1 is outputted.</td></tr>
<tr><td>17</td><td>Threshold Enable: This bit is multiplexed from "Report Trigger Threshold Enable-1" or "Report Trigger Threshold Enable-2" based on the "Internal Report Trigger-1" or "Internal Report Trigger-2" asserted in the Report Reason respectively. "Internal Report</td></tr>
</table></td></tr>
</table>

Field	Description	
		Trigger-1" is given priority over "Internal Report Trigger-2". By default "Report Trigger Threshold Enable-1" is outputted.
	16	**Timer Enabled**
	15:0	Reserved

Performance Counting Register Interface

Global Registers
OACTXID - Observation Architecture Control Context ID
OA_IMR - OA Interrupt Mask Register
OASTATUS - Observation Architecture Status Register
OAHEADPTR - Observation Architecture Head Pointer
OATAILPTR - Observation Architecture Tail Pointer
OABUFFER - Observation Architecture Buffer
OASTARTTRIG_COUNTER - Observation Architecture Start Trigger Counter
OARPTTRIG_COUNTER - Observation Architecture Report Trigger Counter
OAREPORTTRIG2 - Observation Architecture Report Trigger 2
OAREPORTTRIG6 - Observation Architecture Report Trigger 6
CEC0-0 - Customizable Event Creation 0-0
CEC1-0 - Customizable Event Creation 1-0
CEC1-1 - Customizable Event Creation 1-1
CEC2-0 - Customizable Event Creation 2-0
CEC2-1 - Customizable Event Creation 2-1
CEC3-0 - Customizable Event Creation 3-0
CEC3-1 - Customizable Event Creation 3-1
CEC4-0 - Customizable Event Creation 4-0
CEC5-0 - Customizable Event Creation 5-0
CEC5-1 - Customizable Event Creation 5-1
CEC6-0 - Customizable Event Creation 6-0
CEC6-1 - Customizable Event Creation 6-1
CEC7-0 - Customizable Event Creation 7-0
CEC7-1 - Customizable Event Creation 7-1
EU_PERF_CNT_CTL0 - Flexible EU Event Control 0
EU_PERF_CNT_CTL1 - Flexible EU Event Control 1
EU_PERF_CNT_CTL2 - Flexible EU Event Control 2
EU_PERF_CNT_CTL3 - Flexible EU Event Control 3
EU_PERF_CNT_CTL4 - Flexible EU Event Control 4
EU_PERF_CNT_CTL5 - Flexible EU Event Control 5
EU_PERF_CNT_CTL6 - Flexible EU Event Control 6

Symmetrical Registers
OAPERF_A0 - Aggregate Perf Counter A0
OAPERF_A0_UPPER - Aggregate Perf Counter A0 Upper DWord
OAPERF_A1 - Aggregate Perf Counter A1
OAPERF_A1_UPPER - Aggregate Perf Counter A1 Upper DWord
OAPERF_A2 - Aggregate Perf Counter 2
OAPERF_A2_UPPER - Aggregate Perf Counter A2 Upper DWord
OAPERF_A3 - Aggregate Perf Counter A3
OAPERF_A3_UPPER - Aggregate Perf Counter A3 Upper DWord
OAPERF_A4 - Aggregate Perf Counter A4
OAPERF_A4_UPPER - Aggregate Perf Counter A4 Upper DWord
OAPERF_A4_LOWER_FREE - Aggregate Perf Counter A4 Lower DWord Free
OAPERF_A4_UPPER_FREE - Aggregate Perf Counter A4 Upper DWord Free
OAPERF_A5 - Aggregate Perf Counter A5
OAPERF_A5_UPPER - Aggregate Perf Counter A5 Upper DWord
OAPERF_A6 - Aggregate Perf Counter A6
OAPERF_A6_UPPER - Aggregate Perf Counter A6 Upper DWord
OAPERF_A6_LOWER_FREE - Aggregate Perf Counter A6 Lower DWord Free
OAPERF_A6_UPPER_FREE - Aggregate Perf Counter A6 Upper DWord Free
OAPERF_A7 - Aggregate Perf Counter A7
OAPERF_A7_- Upper Aggregate Perf Counter A7 Upper DWord
OAPERF_A8 - Aggregate Perf Counter A8
OAPERF_A8_UPPER - Aggregate Perf Counter A8 Upper DWord
OAPERF_A9 - Aggregate Perf Counter A9
OAPERF_A9_UPPER - Aggregate Perf Counter A9 Upper DWord
OAPERF_A10 - Aggregate Perf Counter A10
OAPERF_A10_UPPER - Aggregate Perf Counter A10 Upper DWord
OAPERF_A11 - Aggregate Perf Counter A11
OAPERF_A11_UPPER - Aggregate Perf Counter A11 Upper DWord
OAPERF_A12 - Aggregate Perf Counter A12
OAPERF_A12_UPPER - Aggregate Perf Counter A12 Upper DWord
OAPERF_A13 - Aggregate Perf Counter A13
OAPERF_A13_UPPER - Aggregate Perf Counter A13 Upper DWord
OAPERF_A14 - Aggregate Perf Counter A14
OAPERF_A14_UPPER - Aggregate Perf Counter A14 Upper DWord
OAPERF_A15 - Aggregate Perf Counter A15
OAPERF_A15_UPPER - Aggregate Perf Counter A15 Upper DWord
OAPERF_A16 - Aggregate Perf Counter A16

Symmetrical Registers
OAPERF_A16_UPPER - Aggregate Perf Counter A16 Upper DWord
OAPERF_A17 - Aggregate Perf Counter A17
OAPERF_A17_UPPER - Aggregate Perf Counter A17 Upper DWord
OAPERF_A18 - Aggregate Perf Counter A18
OAPERF_A18_UPPER - Aggregate Perf Counter A18 Upper DWord
OAPERF_A19 - Aggregate Perf Counter A19
OAPERF_A19_UPPER - Aggregate Perf Counter A19 Upper DWord
OAPERF_A19_LOWER_FREE - Aggregate Perf Counter A19 Lower DWord Free
OAPERF_A19_UPPER_FREE - Aggregate Perf Counter A19 Upper DWord Free
OAPERF_A20 - Aggregate Perf Counter A20
OAPERF_A20_UPPER - Aggregate Perf Counter A20 Upper DWord
OAPERF_A20_UPPER_FREE - Aggregate Perf Counter A20 Upper DWord Free
OAPERF_A20_LOWER_FREE - Aggregate Perf Counter A20 Lower DWord Free
OAPERF_A21 - Aggregate Perf Counter A21
OAPERF_A21_UPPER - Aggregate Perf Counter A21 Upper DWord
OAPERF_A22 - Aggregate Perf Counter A22
OAPERF_A22_UPPER - Aggregate Perf Counter A22 Upper DWord
OAPERF_A23 - Aggregate Perf Counter A23
OAPERF_A23_UPPER - Aggregate Perf Counter A23 Upper DWord
OAPERF_A24 - Aggregate Perf Counter A24
OAPERF_A24_UPPER - Aggregate Perf Counter A24 Upper DWord
OAPERF_A25 - Aggregate Perf Counter A25
OAPERF_A25_UPPER - Aggregate Perf Counter A25 Upper DWord
OAPERF_A26 - Aggregate Perf Counter A26
OAPERF_A26_UPPER - Aggregate Perf Counter A26 Upper DWord
OAPERF_A27 - Aggregate Perf Counter A27
OAPERF_A27_UPPER - Aggregate Perf Counter A27 Upper DWord
OAPERF_A28 - Aggregate Perf Counter A28
OAPERF_A28_UPPER - Aggregate Perf Counter A28 Upper DWord
OAPERF_A29 - Aggregate Perf Counter A29
OAPERF_A29_UPPER - Aggregate Perf Counter A29 Upper DWord
OAPERF_A30 - Aggregate Perf Counter A30
OAPERF_A30_UPPER - Aggregate Perf Counter A30 Upper DWord
OAPERF_A31 - Aggregate_Perf_Counter_A31
OAPERF_A31_UPPER - Aggregate Perf Counter A31 Upper DWord
OAPERF_A32 - Aggregate_Perf_Counter_A32
OAPERF_A33 - Aggregate_Perf_Counter_A33
OAPERF_A34 - Aggregate_Perf_Counter_A34

Symmetrical Registers
OAPERF_A35 - Aggregate_Perf_Counter_A35
GPU_TICKS - GPU_Ticks_Counter

OA Interrupt Control Registers

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

Bit	Description
31:29	**Reserved. MBZ:** These bits may be assigned to interrupts on future products/steppings.
28	**Performance Monitoring Buffer Half-Full Interrupt:** For internal trigger (timer based) reporting, if the report buffer crosses the half full limit, this interrupt is generated.
27:0	**Reserved: MBZ (These bits must be never set by OA, these bits could be allocated to some other unit)**

- **WDBoxOAInterrupt Vector**
- IMR
- Bit Definition for Interrupt Control Registers

Performance Counter Reporting

When either the MI_REPORT_PERF_COUNT command is received or the internal report trigger logic fires, a snapshot of the performance counter values is written to memory. The format used by HW for such reports is selected using the Counter Select field within the register. The organization and number of report formats vary per project.

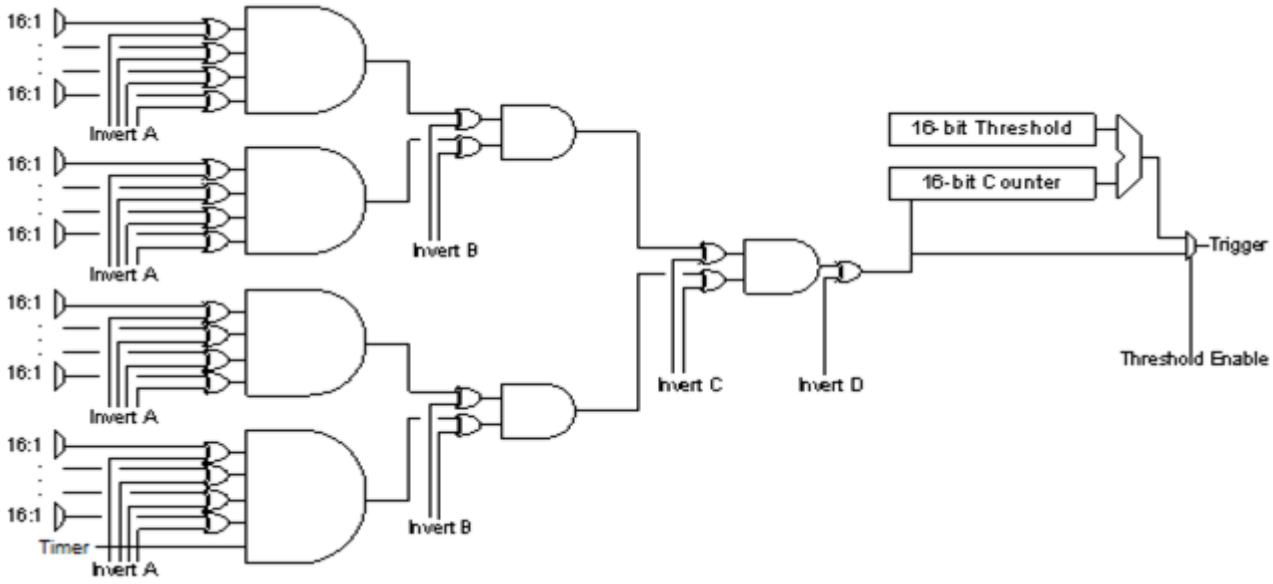
Details of Start Trigger Behavior

- All counters not explicitly defined as free running will advance after the start trigger conditions are met.
- Counting will continue after the start trigger has fired until OA is disabled or device is reset.
- Multiple start triggering blocks (where implemented) are OR'd together in order to allow specification of multiple trigger conditions.
- Bit 18 in the report format reflects whether the start trigger has fired or not.

While architectural intent was that Start Trigger logic would control all qualified counter types (A/B/C), there is a long-standing implementation bug whereby start trigger logic only affects B/C counters.

Configuration of Trigger Logic

OA contains logic to control when performance counter values are reported to memory. This functionality is controlled using the OA report trigger and OA start trigger registers. More detailed register descriptions are included in the Hardware Programming interface. The block diagram below illustrates the logic these registers control.



Note that counters which are 40 bits wide are split in the report format into low DWORD and high byte chunks for simplicity of HW implementation as well as SW-friendly alignment of report data. The performance counter read logically done before writing out report data for these 40-bit counters is guaranteed to be an atomic operation, the counter data is simply swizzled as it is being packed into the report.

Context Switch Triggered Reports

A context load/switch on RCS will cause a performance counter snapshot to be written to memory at the next location in the OA circular report buffer using the perf counter format selected in OACONTROL (). This functionality can be leveraged when preemption is enabled to re-construct the contribution of a specific context to a performance counter delta, requires SW to consider both the delta reported by MI_REPORT_PERF and the reports that may have been issued to OABUFFER by intervening contexts.

A context load/switch on RCS will cause a performance counter snapshot to be written to memory at the next location in the OA circular report buffer using the perf counter format selected in **OARCONTROL**. This functionality can be leveraged when preemption is enabled to re-construct the contribution of a specific context to a performance counter delta, requires SW to consider both the delta reported by MI_REPORT_PERF and the reports that may have been issued to OABUFFER by intervening contexts.

Frequency Change Triggered Reports

A GFX frequency change will cause a performance counter snapshot to be written to memory at the next location in the OA circular report buffer using the perf counter format selected in OACONTROL (). Please note that a change back to the same frequency can occur and that such changes will still cause a performance counter report to occur.

A GFX frequency change will cause a performance counter snapshot to be written to memory at the next location in the OA circular report buffer using the perf counter format selected in **OARCONTROL**. Please note that a change back to the same frequency can occur and that such changes will still cause a performance counter report to occur.

Aggregating Counters

The table below described the desired high-level functionality from each of the aggregating counters.

Note that there is no counter of 2x2s sent to pixel shader, this is based on the assumption that the pixel shader invocation pipeline statistics counter increments for partially lit 2x2s as well and hence does not require a duplicate performance counter.

Please also note that some of the information provided by A-counters is useful for GFX/system load-balancing and is hence made available via free-running counters which do not require initial setup and count irrespective of OA enable/disable or freeze.

Counter #	Event	Description
A0	GPU Busy	GPU is not idle (includes all GPU engines). Link to detailed register definition:
A1	# of Vertex Shader Threads Dispatched	Count of VS fused threads dispatched to EUs Link to detailed register definition:
A2	# of Hull Shader Threads Dispatched	Count of HS fused threads dispatched to EUs Link to detailed register definition:
A3	# of Domain Shader Threads Dispatched	Count of DS fused threads dispatched to EUs Link to detailed register definition:
A4	# of GPGPU Threads Dispatched	Count of GPGPU fused threads dispatched to EUs. Available on both qualified and free-running counters. Link to detailed register definition:
A5	# of Geometry Shader Threads Dispatched	Count of GS fused threads dispatched to EUs Link to detailed register definition:
A6	# of Pixel Shader Threads Dispatched	Count of PS fused threads dispatched to EUs. Available on both qualified and free-running counters. Link to detailed register definition:
A7	Aggregating EU counter 0	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition:
A8	Aggregating EU counter 1	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition:
A9	Aggregating EU counter 2	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition:
A10	Aggregating EU counter 3	User-defined (details in Flexible EU Event Counters section)

Counter #	Event	Description
		Link to detailed register definition:
A11	Aggregating EU counter 4	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition:
A12	Aggregating EU counter 5	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition:
A13	Aggregating EU counter 6	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition:
A14	Aggregating EU counter 7	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition:
A15	Aggregating EU counter 8	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition:
A16	Aggregating EU counter 9	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition:
A17	Aggregating EU counter 10	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition:
A18	Aggregating EU counter 11	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition:
A19	Aggregating EU counter 12	Available on both qualified and free-running counters User-defined (details in Flexible EU Event Counters section) Link to detailed register definition:
A20	Aggregating EU counter 13	Available on both qualified and free-running counters User-defined (details in Flexible EU Event Counters section) Link to detailed register definition:
A21	2x2s Rasterized	Count of the number of samples of 2x2 pixel blocks generated from the input geometry before any pixel-level tests have been applied. (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.)

Counter #	Event	Description
		Link to detailed register definition:
A22	2x2s Failing Fast pre-PS Tests	Count of the number of samples failing fast "early" (i.e. before pixel shader execution) tests (counted at 2x2 granularity). (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.) Link to detailed register definition:
A23	2x2s Failing Slow pre-PS Tests	Count of the number of samples of failing slow "early" (i.e. before pixel shader execution) tests (counted at 2x2 granularity). (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.) Link to detailed register definition:
A24	2x2s Killed in PS	Number of samples entirely killed in the pixel shader as a result of explicit instructions in the kernel (counted in 2x2 granularity). (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.) Link to detailed register definition:
A25	2x2s Failing post-PS Tests "POSTPS_DEPTH_STENCIL_ALPHA_FAIL"	Number of samples that entirely fail "late" tests (i.e. tests that can only be performed after pixel shader execution). Counted at 2x2 granularity. (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.) Link to detailed register definition:
A26	2x2s Written To Render Target	Number of samples that are written to render target.(counted at 2x2 granularity). MRT case will report multiple writes per 2x2 processed by the pixel shader. (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.) Please note that this counter will not advance if a render target update does not occur and that pixel masking operations performed by the fixed function HW or shader may not be reflected in counters A22-A25 which only track their specific defined operations. This can lead to an apparent discrepancy between A21 vs. A22-A25 vs. A26/A27. Link to detailed register definition:
A27	Blended 2x2s Written to Render Target	Number of samples of blendable that are written to render

Counter #	Event	Description
		target.(counted at 2x2 granularity). MRT case will report multiple writes per 2x2 processed by the pixel shader. (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.) Please note that this counter will not advance if a render target update does not occur and that pixel masking operations performed by the fixed function HW or shader may not be reflected in counters A22-A25 which only track their specific defined operations. This can lead to an apparent discrepancy between A21 vs. A22-A25 vs. A26/A27. Link to detailed register definition:
A28	2x2s Requested from Sampler	Aggregated total 2x2 texel blocks requested from all EUs to all instances of sampler logic. Link to detailed register definition:
A29	Sampler L1 Misses	Aggregated misses from all sampler L1 caches. Please note that the number of L1 accesses varies with requested filtering mode and in other implementation specific ways. Hence it is not possible in general to draw a direct relationship between A28 and A29. However, a high number of sampler L1 misses relative to texel 2x2s requested frequently degrades sampler performance. Link to detailed register definition:
A30	SLM Reads	Total read requests from an EU to SLM (including reads generated by atomic operations). Link to detailed register definition:
A31	SLM Writes	Total write requests from an EU to SLM (including writes generated by atomic operations). Link to detailed register definition:
A32	Other Shader Memory Accesses	Aggregated total requests from all EUs to memory surfaces other than render target or texture surfaces (e.g. shader constants). Link to detailed register definition:
A33	Other Shader Memory Accesses That Miss First-Level Cache	Aggregated total requests from all EUs to memory surfaces other than render target or texture surfaces (e.g. shader constants) that miss first-level cache. Link to detailed register definition:
A34	Atomic Accesses	Aggregated total atomic accesses from all EUs. This counter

Counter #	Event	Description
		increments on atomic accesses to both SLM and URB. Link to detailed register definition: **Workaround** SLM atomics are not included previously by this OA event (only global memory atomics are counted), a workaround using B/C counters is possible.
A35	Barrier Messages	Aggregated total kernel barrier messages from all Eus (one per thread in barrier). Link to detailed register definition:

SPM Counters

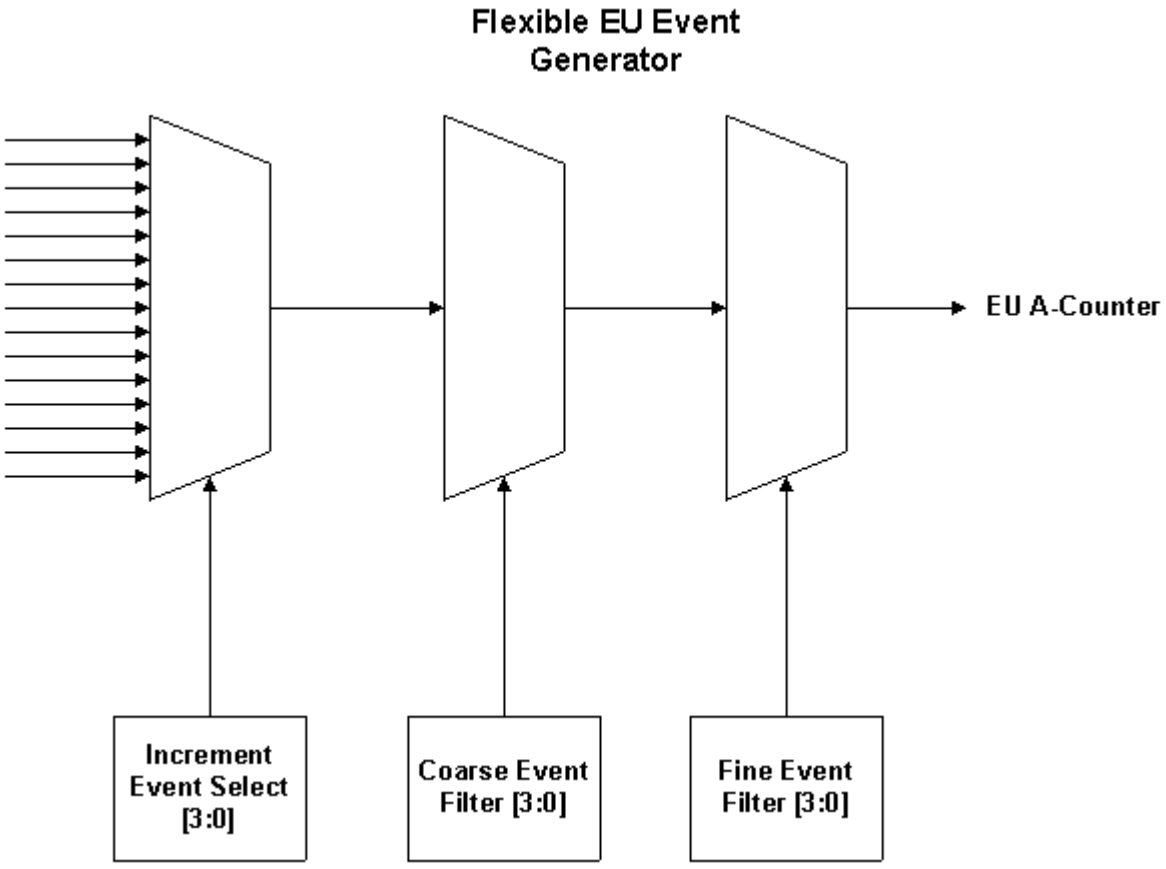
Counter #	Event	Description
SPM0	EU Stall	Event reflects the condition where an EU is not idle but also not processing an ISA instruction. Each increment of the event reflects 4 clocks where a single EU met this condition.
SPM1	EU IPC	Event counts the number of ISA FPU/EM instructions that GT processes. It comprehends cases where multiple instructions are processed in a single clock. Each increment of this event reflects 8 instructions processed.
SPM2	Threads loaded	Event counts the total number of threads that have been fully loaded onto an EU in a given clock. This event DOES NOT include the time where the thread header is being sent to the EU. The per-clock increment is added to an accumulator each clock, a single increment of this event reflects that the accumulator has reached 8.
SPM3	EU Not Idle	Event reflects the condition where an EU is not idle. Each increment of the event reflects 8 clocks where a single EU met this condition.
SPM4	Sampler Not Idle	Event counts sampler activity
SPM5	EU Stalled & Sampler Not Idle	Event reflects the condition where the EU has sent a request(s) to sampler and all threads on the EU are stalled. Please note that the EU could be stalled for reasons other than sampler as well. Each increment of the event reflects 8 clocks where a single EU met this condition.

Flexible EU Event Counters

Since EU performance events are most interesting in many cases when aggregated across all EUs and many interesting EU performance events are limited to certain APIs (e.g. hull shader kernel stats only applicable when running a DX11+ workload), adds some additional flexibility to the aggregated counters coming from the EU array.

The following block diagram shows the high-level flow that generates each flexible EU event.

Note that no support is provided for differences between flexible EU event programming between EUs because the resulting output from each EU is eventually merged into a single OA counter anyway.



Supported Increment Events

Increment Event	Encoding	Notes
EU FPU Pipeline Active	0b00000	Signal that is high on every EU clock where the EU FPU pipeline is actively executing an ISA instruction.
EU EM Pipeline Active	0b00001	Signal that is high on every EU clock where the EU EM pipeline is actively executing an ISA instruction.
		Only fine event filters 0b0000, 0b0111, 0b1000, 0b1001, and 0b1010 are supported with this increment event.
EU SEND Pipeline Active	0b00010	Signal that is high on every EU clock where the EU send pipeline is actively executing an ISA instruction. Only fine event filters 0b0000,0b0101, 0b0111, 0b1000, 0b1001, and 0b1010 are supported with this increment event.
EU FPU & EM Pipelines Concurrently Active	0b00011	Signal that is high on every EU clock where the EU FPU and EM pipelines are both actively executing an ISA instruction. Only coarse event filters 0b0000, 0b0111, and 0b1000 are supported with this increment event. Only fine event filters 0b0000, 0b0111, 0b1000, 0b1001, and 0b1010 are supported with this increment event.
Some EU Pipeline Active	0b00100	Signal that is high on every EU clock where at least one EU pipeline is actively executing an ISA instruction. Only coarse event filters 0b0000, 0b0111, and 0b1000 are supported with this increment event. Only fine event filters 0b0000,0b0101, 0b0111, 0b1000, 0b1001, and 0b1010 are supported with this increment event.
		Signal that is high on every EU clock where at least one EU pipeline is actively executing an ISA instruction. All coarse event filters are supported. Only fine event filters 0b0000,0b0101, 0b0111, 0b1000, 0b1001, and 0b1010 are supported with this increment event.
At Least 1 Thread Loaded But No EU Pipeline Active	0b00101	Signal that is high on every EU clock where at least one thread is loaded but no EU pipeline is actively executing an ISA instruction. Only coarse event filters 0b0000, 0b0111, and 0b1000 are supported with this increment event. Only fine event filters 0b0000, 0b0111, 0b1000, 0b1001, and 0b1010 are supported with this increment event.
		Signal that is high on every EU clock where at least one thread is loaded but no EU pipeline is actively executing an ISA instruction. All coarse event filters are supported. Only fine event filters 0b0000, 0b0111, 0b1000, 0b1001, and 0b1010 are supported with this increment event.
Threads loaded integrator == max threads for current HW SKU	0b01000	Implies an accumulator which increases every EU clock by the number of loaded threads, signal pulses high for one clock when the accumulator exceeds a multiple of the number of thread slots (e.g. for a 8-thread EU, signal pulses high every clock where the increment causes a 3-bit accumulator to overflow). Only coarse event filters 0b0000, 0b0111, and 0b1000 are supported with this increment event. Only fine event filters 0b0000, 0b0111, 0b1000, 0b1001, and 0b1010 are supported with this increment event.
No event	0b01111	Expected HW default, allows logic to be power-optimized.

Supported Coarse Event Filters

Coarse Event Filter	Encoding	Notes
No mask	0b0000	Never masks increment event.
VS Thread Filter	0b0001	For increment events 0b00000/0b00001/0b00010, masks increment events unless the FFID which dispatched the currently executing thread equals FFID of VS.
		For increment events 0b00100/0b00101, masks increment event unless at least one of the loaded threads was dispatched by VS.
HS Thread Filter	0b0010	For increment events 0b00000/0b00001/0b00010, masks increment event unless the FFID which dispatched the currently executing thread equals FFID of HS.
		For increment events 0b00100/0b00101, masks increment event unless at least one of the loaded threads was dispatched by HS
DS Thread Filter	0b0011	For increment events 0b00000/0b00001/0b00010, masks increment event unless the FFID which dispatched the currently executing thread equals FFID of DS.
		For increment events 0b00100/0b00101, masks increment event unless at least one of the loaded threads was dispatched by DS.
GS Thread Filter	0b0100	For increment events 0b00000/0b00001/0b00010, masks increment event unless the FFID which dispatched the currently executing thread equals FFID of GS.
		For increment events 0b00100/0b00101, masks increment event unless at least one of the loaded threads was dispatched by GS.
PS Thread Filter	0b0101	For increment events 0b00000/0b00001/0b00010, masks increment event unless the FFID which dispatched the currently executing thread equals FFID of PS.
		For increment events 0b00100/0b00101, masks increment event unless at least one of the loaded threads was dispatched by PS.
TS Thread Filter	0b0110	For increment events 0b00000/0b00001/0b00010, masks increment event unless the FFID which dispatched the currently executing thread equals FFID of TS.
		For increment events 0b00100/0b00101, masks increment event unless at least one of the loaded threads was dispatched by TS.
Row = 0	0b0111	Masks increment event unless the row ID for this EU is 0 (control register is in TDL so only have to check within quarter-slice).

Fine Event Filters

Fine Event Filter	Encoding	Notes
None	0b0000	Never mask increment event.
Cycles where hybrid instructions are being executed	0b0001	Masks increment event unless the instruction(s) being executed on the pipeline(s) selected by the increment event are hybrid instructions.
Cycles where ternary instructions are being executed	0b0010	Masks increment event unless the instruction(s) being executed on the pipeline(s) selected by the increment event are ternary instructions.
Cycles where binary instructions are being executed	0b0011	Masks increment event unless the instruction(s) being executed on the pipeline(s) selected by the increment event are binary instructions.
Cycles where mov instructions are being executed	0b0100	Masks increment event unless the instruction(s) being executed on the pipeline(s) selected by the increment event are mov instructions.
Cycles where sends start being executed	0b0101	Masks increment event unless the instruction(s) being executed on the pipeline(s) selected by the increment event are send start of dispatch. Note that if this fine event filter is used in combination with increment events not related to the EU send pipeline (e.g. FPU0 active), the associated flexible event counter will increment in an implementation-specific manner.
EU# = 0b00	0b0111	Masks increment event unless the EU number for this EU is 0b00.
EU# = 0b01	0b1000	Masks increment event unless the EU number for this EU is 0b01.
EU# = 0b10	0b1001	Masks increment event unless the EU number for this EU is 0b10.
EU# = 0b11	0b1010	Masks increment event unless the EU number for this EU is 0b11.

Flexible EU Event Config Registers

EU_PERF_CNT_CTL0 - Flexible EU Event Control 0

EU_PERF_CNT_CTL1 - Flexible EU Event Control 1

EU_PERF_CNT_CTL2 - Flexible EU Event Control 2

EU_PERF_CNT_CTL3 - Flexible EU Event Control 3

EU_PERF_CNT_CTL4 - Flexible EU Event Control 4

EU_PERF_CNT_CTL5 - Flexible EU Event Control 5

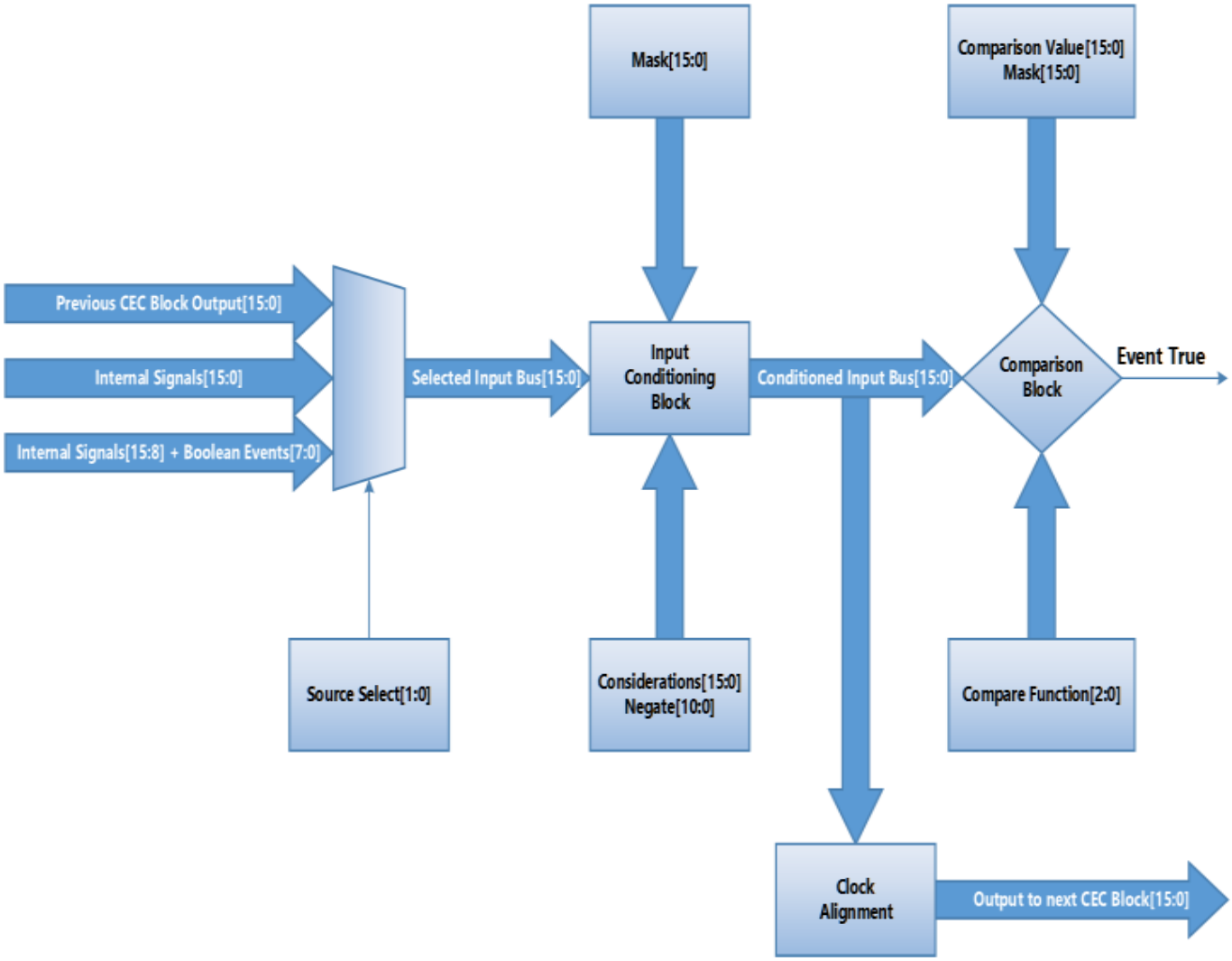
EU_PERF_CNT_CTL6 - Flexible EU Event Control 6

Custom Event Counters

Also known as B-counters, the events counted in these counters are defined from Boolean combinations of

input signals using the custom event creation logic built into OA.

The following diagram(s) illustrate(s) the structure used to create a custom event. Every B-counter has such a block.


Mask[15:0]
Comparison Value[15:0]
Mask[15:0]
Previous CEC Block Output[15:0]
Internal Signals[15:0]
Internal Signals[15:8] + Boolean Events[7:0]
Selected Input Bus[15:0]
Input Conditioning Block
Conditioned Input Bus[15:0]
Comparison Block
Event True
Source Select[1:0]
Considerations[15:0]
Negate[10:0]
Compare Function[2:0]
Clock Alignment
Output to next CEC Block[15:0]

Interrupts

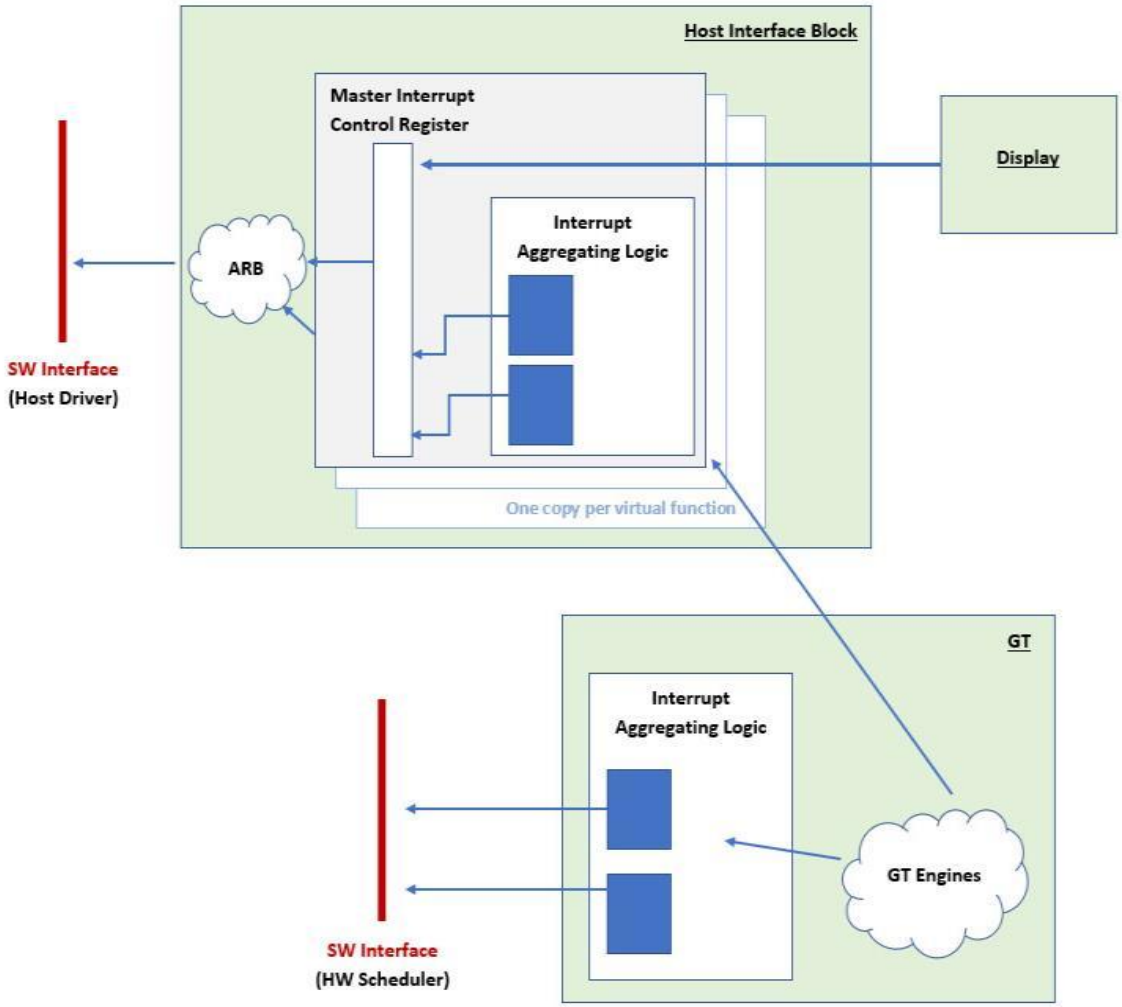
This is the Interrupts section.

Overview:

The Graphics device is comprised of a number of independent engines that can be invoked to execute workloads. Engines communicate status primarily through interrupts. The Graphics device supports two models of scheduling and handling of interrupts:

- Host SW schedules and manages all interrupts
- Scheduling and related interrupts are managed by hardware scheduler (MinIA micro-controller) and host SW manages interrupts not related to scheduling.

The hardware can be configured to work in either of these models. HW scheduling is the preferred mode because it provides best utilization of resources. The figure below shows the high-level overview of the interrupt infrastructure.



The interrupt infrastructure is designed to support both of these models. Each engine is allocated a set of interrupt bits that it can set to report events (the number of bits allotted to each engine varies -- most engines are allocated 16bits, some engines which have more events are allocated 32bits). Interrupt messages sent by engines result in interrupt bits being recorded in MMIO registers and an interrupt being generated to the servicing agent (MinIA scheduler or Host SW). The interrupt handler determines the source of the interrupt (by reading registers) and then processes the interrupts. Processing interrupts involves reading the interrupt status register, performing the operations for handling the interrupt and indicating completion of handling by writing to registers (clear).

When using the HW scheduler, the scheduling related interrupts are directed to the MinIA scheduler.

GT Engine Interrupts:

Within GT, engines are categorized into different engine classes and instances. An engine class is used to differentiate between engines that perform different functions (Copy, Render, VideoDecode, VideoEncode, etc). A product may have a number of instances of a specific engine class e.g.: GT2 has 2 instances of VD, GT3 has 4 instances of VD, etc. The following table lists various engine classes as well as instances within each class.

Engine Class	Engine Instance Name	ClassID[2:0]	InstanceID[5:0]
Render	RCS	0	0
Video Decode	VCS0-N	1	0-N
Video Enhancement Engine	VECS0-N/2	2	0-N/2
Copy Engine	BCS	3	0
	GTPM	4	1
	WDOAPerf	4	2
	SCTRG	4	3
	KCR	4	4
	Gunit	4	5
	CSME	4	6
Compute Engine	CCS0-N	5	0-N
Reserved		6-7	

Each engine reports up to 16 interrupts to interrupt handling logic. Source identification data is included in interrupt messages to interrupt aggregating logic, i.e. when reporting an interrupt to either host or graphics firmware, the generating engine must identify itself. 16 bits of identification is sent along with interrupt data, and comprises Engine Class ID, Instance ID and Virtual Function Number. Interrupt bit definition varies per engine class, these are listed in the Global section.

Format of interrupt message:

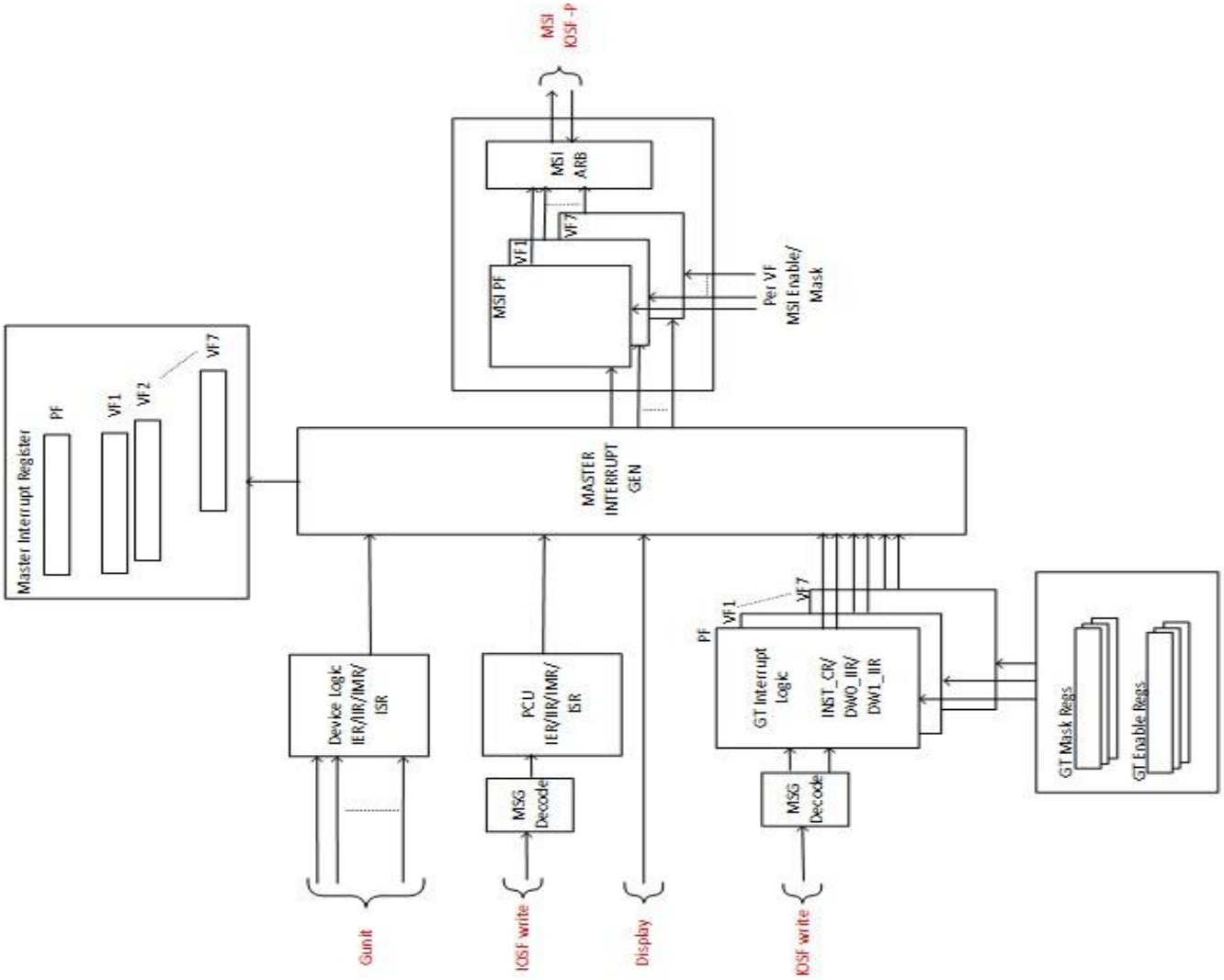
Bit Fied	Purpose
[31:30]	Reserved
[29:27]	VF ID
[26]	Reserved
[25:20]	Instance ID
[19]	Reserved
[18:16]	Engine Class ID
[15:0]	Interrupt data

Hardware Scheduler/MinIA SW Interface

Graphics interrupts to scheduling firmware are delivered as two unique vector values. Each vector accounts for 32 graphics engines. Firmware processes each of two groups of graphics engines independently.

Service routines are independent for the two interrupt vectors presented to the MinIA firmware.

The diagrams below assume SRIOV-8 implementation which does not include memory-based interrupt support.



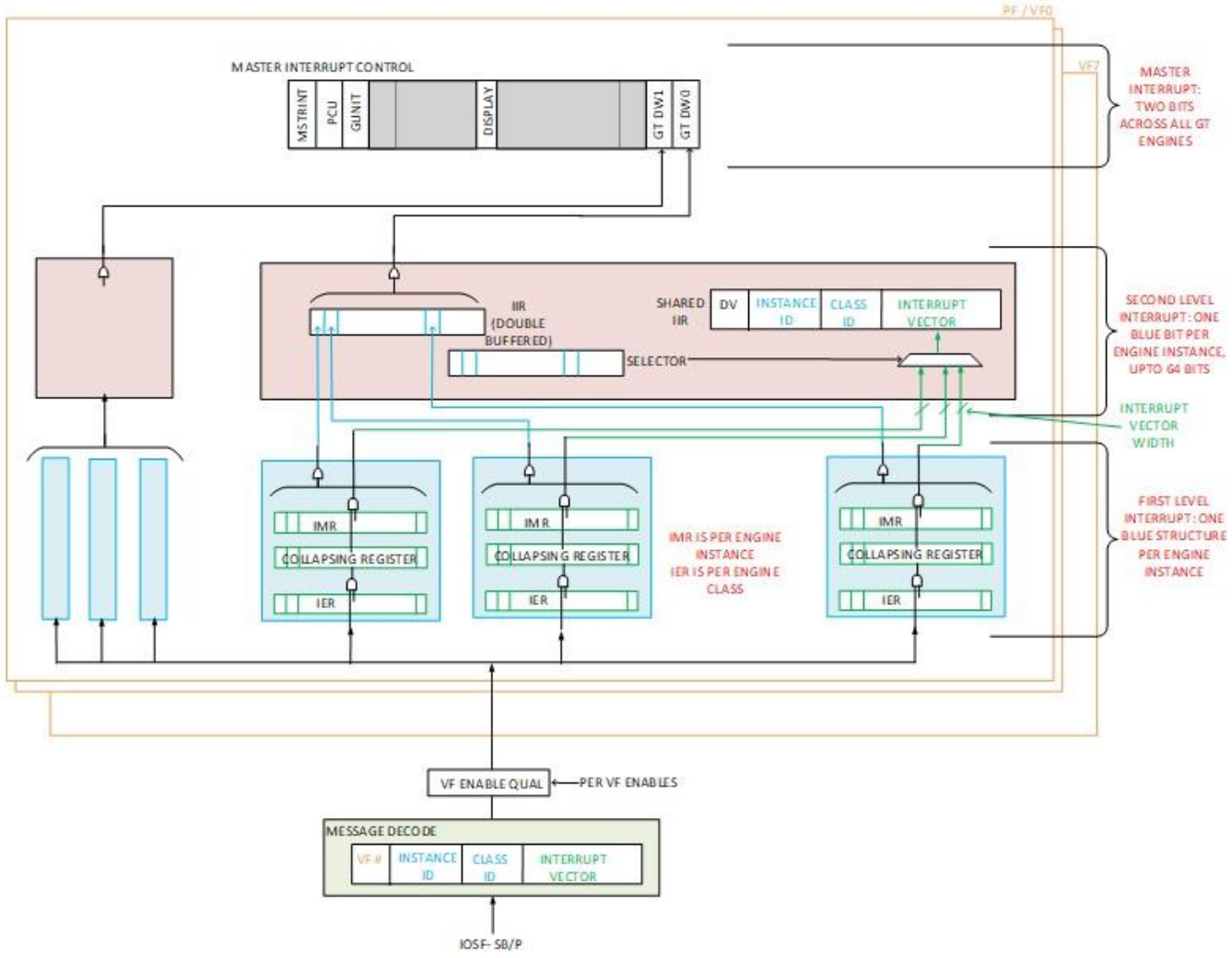
Interrupts to Host are delivered via a Master Interrupt Control Register. Graphics interrupts use 2 bits in the Master Interrupt Control Register. In addition, interrupt events from Display are also represented in the Master Interrupt Control Register. Multiple copies of Master Interrupt Control Register exist, one for every virtual machine in the system.

Interrupt bits in the Master Interrupt Control Register are Read-Only bits, and are level indications that a second level interrupt is present (As seen earlier, second level interrupts per client are OR-ed together to set a bit in the Master. When the second level IIR is cleared, the bit represented in the Master will be 0.). An interrupt is sent to driver whenever bits are set in the Master Interrupt Control Register and the Enable bit is also set.

As a result of this interrupt, SW first resets the Master Control Enable bit. SW then reads the Master Interrupt Control register into a local variable, and works off this local variable to service interrupts. Once all lower level interrupts have been serviced, SW writes the Master Interrupt Control register to set the Master Control Enable bit.

Interrupt Aggregating Logic

A hierarchical interrupt status infrastructure is provided to efficiently determine the source of the interrupt. The first level of interrupts is generated by GT Engines. Interrupt handling logic accumulates these interrupts from the various engines, and organizes it as a single bit per engine in a second level. 32 bits of second level interrupts are OR-ed together to generate a DW-level interrupt event for up to 32 engines. Two such events are used to provide support for up to 64 GT engines. When communicating with the MinIA, these events are mapped to two unique interrupt vectors in the MinIA LAPIC. When communicating with host driver, these events form two bits of the Master Interrupt Control Register as marked in the picture.



First Level Interrupt Bits:

When an interrupt event comes into the interrupt handling logic, it is AND-ed with a per-Engine Enable register (IER). Only enabled events make forward progress. Disabled events are simply dropped by the interrupt handling logic. [Note that multiple instances of the same engine type (except those in the 'Other' Engine Class) share the same Enable register.]

Enabled interrupts are logged in a per-instance, non-SW readable Collapsing Register. These events are AND-ed with (the inverse of) a per-Instance Mask Register (IMR). Only unmasked events make forward

progress. Masked events remain in the per-Instance Collapsing Register until they are unmasked. [Note that every instance (even of the same engine type) has its own Mask Register.]

Unmasked events in the per-Instance Collapsing Register are OR-ed together to produce a single second level interrupt event.

Second Level Interrupt Bits:

Second level interrupt events are stored in a double buffered IIR structure. A snapshot of events is taken when SW reads the IIR. From the time of read to the time of SW completely clearing the second-level IIR (to indicate end of service), all incoming interrupts are logged in a secondary storage structure. This guarantees that the record of interrupts SW is servicing will not change while under service.

Bits in the second-level IIR are OR-ed together to generate a DW-level event. The IIR is cleared by writing 1s. If events exist in the secondary storage at the time that the IIR is completely cleared, a second DW-level event will be generated.

Shared IIR, Selector:

Shared IIR and Selector registers are used when SW is in the process of handling reported interrupts. As a result of a GT interrupt (DW-level interrupt), SW reads the second-level IIR register. The read provides an indication of engines needing service. SW must then service engines one at a time by writing a one-hot selection into the Selector Register.

When a selection is made by writing the Selector, interrupt handling logic presents all the unmasked interrupt bits (first level interrupt events) for the selected engine in the Shared IIR, and sets the Data-Valid bit (MSB). SW can then read the Shared IIR and take action for the reported events. SW must clear the Shared IIR by writing 1 to the Data-Valid bit to indicate end of service for the selected engine. This clearing of the Shared IIR Data-Valid bit clears both the Shared IIR as well as the Selector. Note that the Selector data must be one-hot. Selector must not have a bit set that is not set in the second-level IIR at the time of SW read.

SW then repeats the above steps for each bit set in the second-level IIR. Multiple rounds of Selector write-Shared IIR clear may be required to service a DW level interrupt a single time.

Second-level IIR bits are cleared only after individual engines are serviced via the Selector write -Shared IIR clear routine. This clearing can be done after each iteration through the Selector write-Shared IIR clear routine (i.e. one second-level bit cleared after each iteration), or all at once after all engines have been serviced. Second-level IIR bits must not be cleared without first servicing that engine's interrupts via the Selector and Shared IIR registers.

Enable and Mask Registers:

Interrupt aggregating logic includes Enable registers(IER) per Engine Class. Different instances of the same engine class use the same Enable register, except for engines in the 'Other' class. Each instance in the 'Other' class has its own Enable register.

Interrupt aggregating logic also includes Mask registers (IMR). Each engine instance, even within the same Engine Class, has a unique Mask Register.

Enables for Engine classes at the two software interfaces are typically complements of each other.