Intel® Iris® Plus Graphics and UHD Graphics Open Source

Programmer's Reference Manual

For the 2019 10th Generation Intel Core™ Processors based on the "Ice Lake" Platform

Volume 13: SW/HW System Interface

January 2020, Revision 1.0

Creative Commons License

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

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

Notices and Disclaimers

Table of Contents

SW/HW System Interface

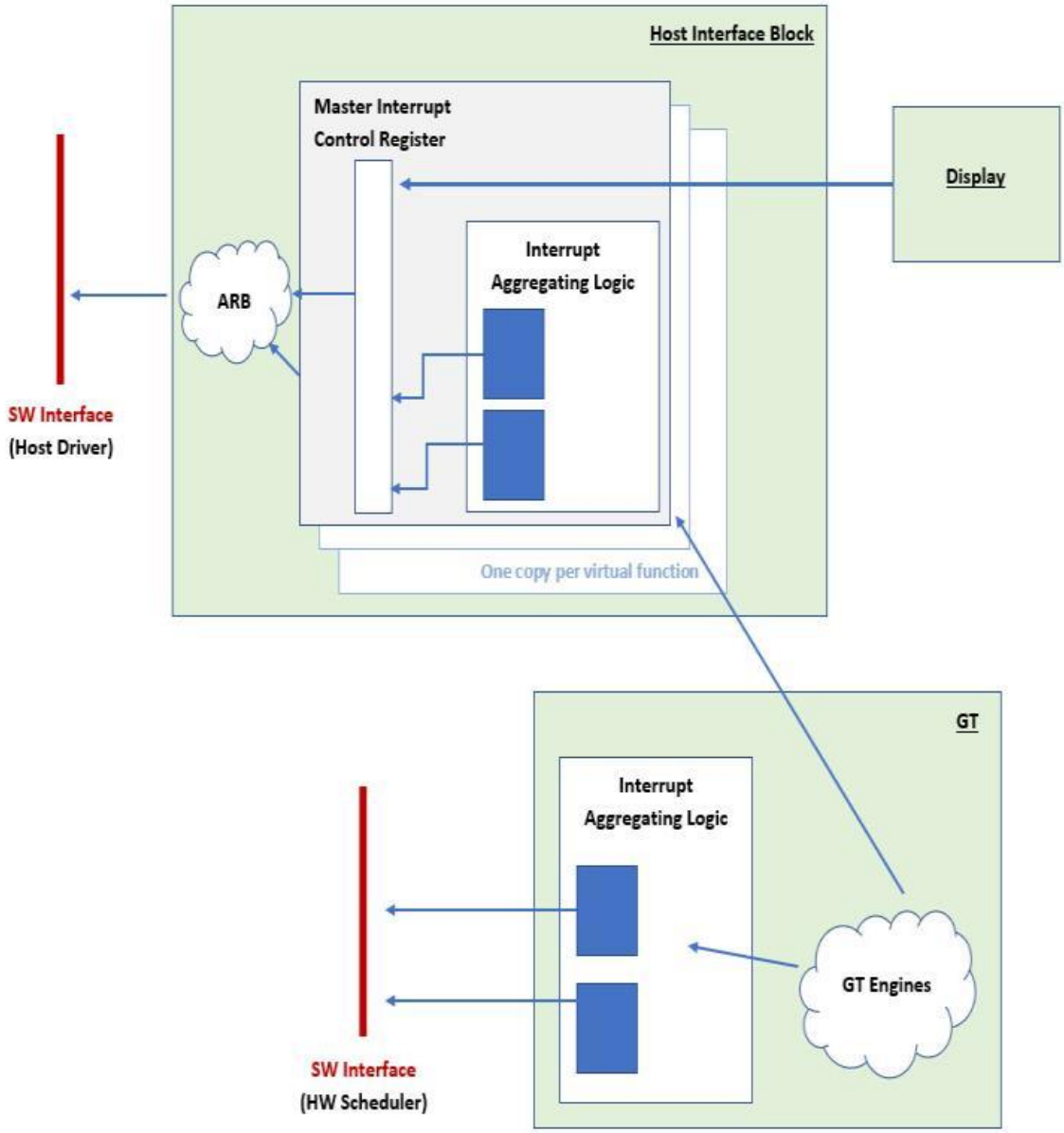
Interrupts

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 Gen 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
Other	GuC	4	0
	GTPM	4	1
	WDOAPerf	4	2
	SCTRG	4	3
	KCR	4	4
	Gunit	4	5
	CSME	4	6
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 Bspec in the Global/ section.

Format of interrupt message:

Bit Fied	Purpose
[31:30]	Reserved
[29:27]	VF ID
[26]	Reserved
[25:20]	Instance ID

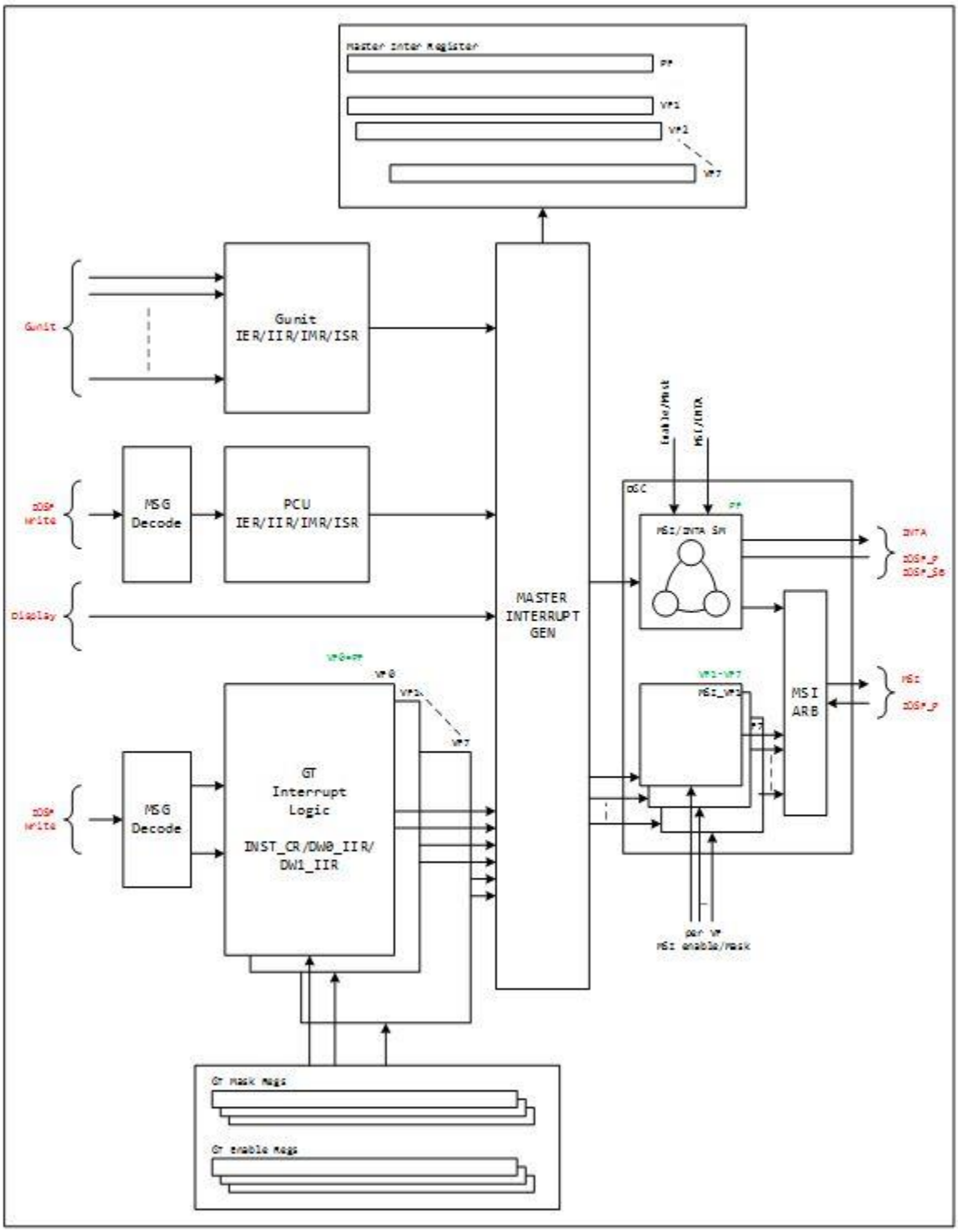
[19]	Reserved
[18:16]	Engine Class ID
[15:0]	Interrupt data

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.

Host SW Interface



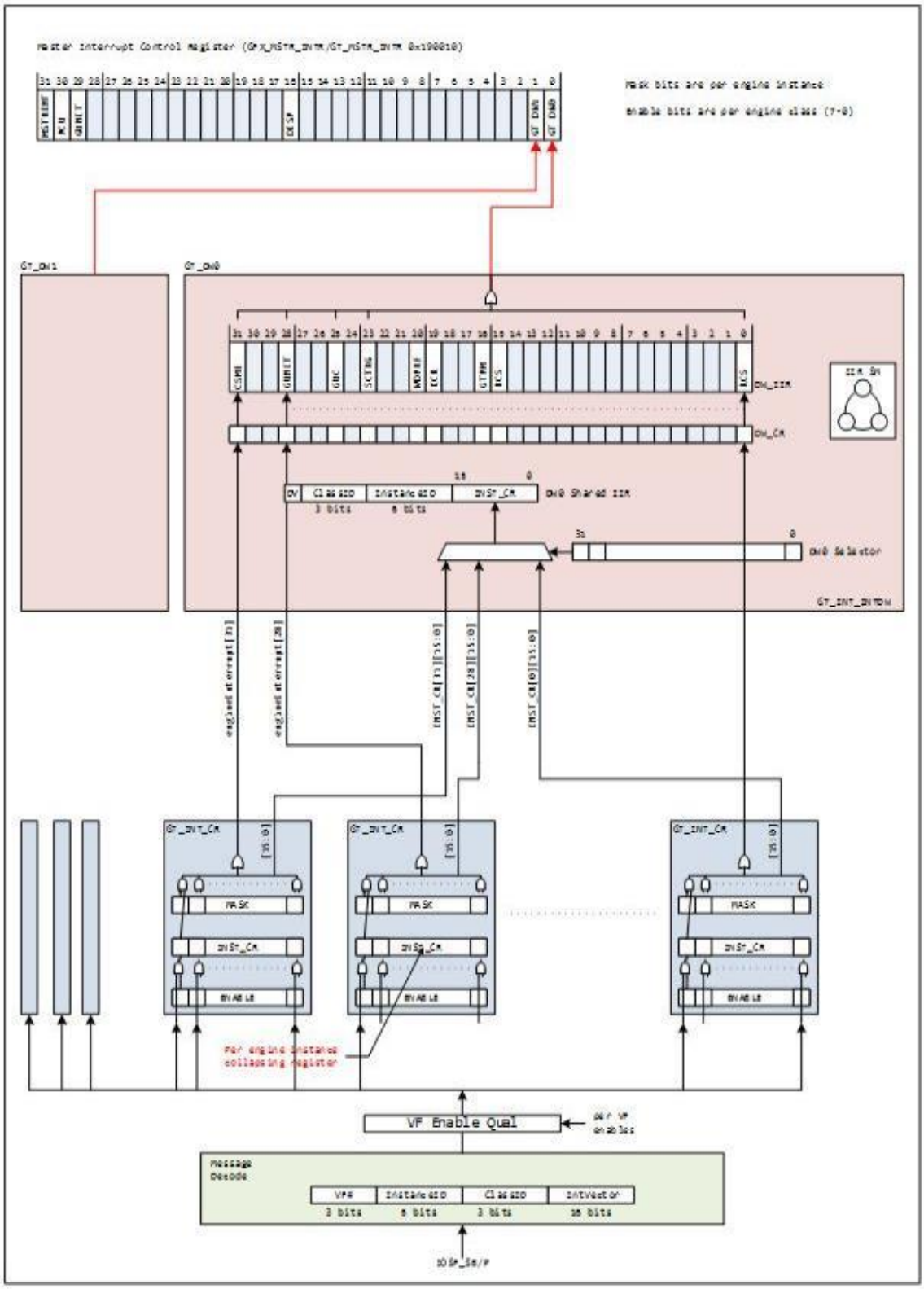
Interrupts to Host are delivered via a Master Interrupt Control Register. Graphics interrupts use 2 bits in the Master Interrupt Control Register. In addition, interrupt events from Display are also represented in the Master Interrupt Control Register. Multiple copies of INT DW and Master Interrupt Control 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. These DW-level interrupt events are marked in red in the picture below. 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.


Master Interrupt Control Register (GFX_MSTR_INTR/GT_MSTR_INTR 0x190010)
Mask bits are per engine instance
Enable bits are per engine class (7:0)
GT_DW1
GT_DW0
GT_INT_INTRW
VF Enable Qual
Message Decode
VFN
Instance
Class
IntVector
3 bits
6 bits
3 bits
16 bits
MASK
INST_CM
ENABLE
Per engine instance collapsing register

Nomenclature:

First level interrupts are stored in storage named '**per-Instance Collapsing Register**' (INST_CR in the picture). These are 16 bits wide, one such storage exists per interrupt producing engine within GT.

Second level interrupts are stored in storage referred to as '**GT INT DW'** (DW_IIR, DW_CR in the picture). These are 32 bits wide. Two such DWs exist, DW0 and DW1

Each GT INT DW is accompanied by two other registers:

Selector: This is a one-hot version of GT INT DW, and indicates which engine of 32 is next to be serviced.

Shared IIR: This presents the 16 bit interrupt data of the engine selected by the Selector for SW to service.

First Level Interrupt Bits:

When an interrupt event comes into the interrupt handling logic, it is AND-ed with a per-Engine Enable register. 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 Collapsing Register. These events are AND-ed with (the inverse of) a per-Instance Mask Register. 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 the GT INT DW. GT INT DW is a double buffered structure. A snapshot of events is taken when SW reads GT INT DW. From the time of read to the time of SW completely clearing GT INT DW (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 GT INT DW_IIR are OR-ed together to generate a DW-level event. INT DW is cleared by writing 1s. If events exist in the secondary storage (DW_CR) at the time that INT DW is completely cleared, a second DW-level event will be generated.

Shared IIR, Selector:

The 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 DW 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 DW_IIR.

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

GT INT DW bits are cleared by SW writing a 1 to these bits. This is done 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 bit in GT INT DW cleared after each iteration), or all at once after all engines have been serviced. DW_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 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. 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.

External Interfaces

Uncore Registers

GFX MMIO – MCHBAR Aperture

Address: 140000h – 147FFFh

140000h - 14FFFFh (CNL, ICL and RKL)

140000h - 15FFFFh (TGL+)

Default Value: Same as MCHBAR

Access: Aligned Word, Dword, or Qword Read

This range defined in the graphics MMIO range is an alias with which graphics driver can read and write registers defined in the MCHBAR MMIO space claimed through Device #0. Attributes for registers defined within the MCHBAR space are preserved when the same registers are accessed via this space. Registers that the graphics driver requires access to are Rank Throttling, GMCH Throttling, Thermal Sensor, etc.

The Alias functions work for MMIO access from the CPU, reads only. In addition, GT is also able to read registers within this range. Writes to the MCHBAR alias are not allowed (writes will have no affect).

Graphics MMIO registers can be accessed through MMIO BARs in the Gfx Device. The aliasing mechanism is turned off if memory access to the corresponding function is turned off via software or in certain power states.

Please refer to applicable EDS documentation for details of this register's format and behavior.

PCI Device Registers

GFX PCI Registers

Address Space PCI: 0/2/0

Address	Symbol	Name
00000h	VID2_0_2_0_PCI	**Vendor Identification**
00002h	DID2_0_2_0_PCI	**Device Identification**
00004h	PCICMD_0_2_0_PCI	**PCI Command**
00006h	PCISTS2_0_2_0_PCI	**PCI Status**
0000Ch	CLS_0_2_0_PCI	**Cache Line Size**
0000Dh	MLT2_0_2_0_PCI	**Master Latency Timer**
0000Eh	HDR2_0_2_0_PCI	**Header Type**
0000Fh	BIST_0_2_0_PCI	**Built In Self Test**
00010h	GTTMMADR_0_2_0_PCI	**Graphics Translation Table Memory Mapped Range Address**
00018h	GMADR_0_2_0_PCI	**Graphics Memory Range Address**
00020h	IOBAR_0_2_0_PCI	**I/O Base Address**
0002Ch	SVID2_0_2_0_PCI	**Subsystem Vendor Identification**
0002Eh	SID2_0_2_0_PCI	**Subsystem Identification**
00030h	ROMADR_0_2_0_PCI	**Video BIOS ROM Base Address**
00034h	CAPPOINT_0_2_0_PCI	**Capabilities Pointer**
0003Ch	INTRLINE_0_2_0_PCI	**Interrupt Line**
0003Dh	INTRPIN_0_2_0_PCI	**Interrupt Pin**
0003Eh	MINGNT_0_2_0_PCI	**Minimum Grant**
0003Fh	MAXLAT_0_2_0_PCI	**Maximum Latency**
00040h	CAPID0_0_2_0_PCI	**Capability Identifier**
00042h	CAPCTRL0_0_2_0_PCI	**Capabilities Control**
00044h	CAPID0_A_0_2_0_PCI	**Capabilities A**
00048h	CAPID0_B_0_2_0_PCI	**Capabilities B**
00050h	MGGC0_0_2_0_PCI	**PCI Mirror of GMCH Graphics Control**
00054h	DEVEN0_0_2_0_PCI	**Mirror of Device Enable**
00058h	DEV2CTL_0_2_0_PCI	**Device 2 Control**

Address	Symbol	Name
00060h	MSAC_0_2_0_PCI	**Multi Size Aperture Control**
00070h	PCIECAPHDR_0_2_0_PCI	**PCI Express Capability Header**
00072h	PCIECAP_0_2_0_PCI	**PCI Express Capability**
00074h	DEVICECAP_0_2_0_PCI	**Device Capabilities**
00078h	DEVICECTL_0_2_0_PCI	**PCI Express Device Control**
0007Ah	DEVICESTS_0_2_0_PCI	**PCI Express Capability Structure**
000ACh	MSI_CAPID_0_2_0_PCI	**Message Signaled Interrupts Capability ID**
000AEh	MC_0_2_0_PCI	**Message Control**
000B0h	MA_0_2_0_PCI	**Message Address**
000B4h	MD_0_2_0_PCI	**Message Data**
000B8h	MSI_MASK_0_2_0_PCI	**MSI Mask Bits**
000BCh	MSI_PEND_0_2_0_PCI	**MSI Pending Bits**
000C0h	BDSM_0_2_0_PCI	**Mirror of Base Data of Stolen Memory**
000C8h	GFX_VTDBAR_LSB_0_2_0_PCI	**GFX_VTDBAR_LSB**
000CCh	GFX_VTDBAR_MSB_0_2_0_PCI	
000D0h	PMCAPID_0_2_0_PCI	**Power Management Capabilities ID**
000D2h	PMCAP_0_2_0_PCI	**Power Management Capabilities**
000D4h	PMCS_0_2_0_PCI	**Power Management Control and Status**
000E0h	SWSMI_0_2_0_PCI	**Software SMI**
000E4h	GSE_0_2_0_PCI	**Graphics System Event**
000E8h	SWSCI_0_2_0_PCI	**Software SCI**
000FCh	ASLS_0_2_0_PCI	**ASL Storage**
00100h	PASID_EXTCAP_0_2_0_PCI	**PASID Extended Capability Header**
00104h	PASID_CAP_0_2_0_PCI	**PASID Capability**
00106h	PASID_CTRL_0_2_0_PCI	**PASID Control**
00200h	ATS_EXTCAP_0_2_0_PCI	**ATS Extended Capability Header**
00204h	ATS_CAP_0_2_0_PCI	**ATS Capability**
00206h	ATS_CTRL_0_2_0_PCI	**ATS Control**
00300h	PR_EXTCAP_0_2_0_PCI	**Page Request Extended Capability Header**
00304h	PR_CTRL_0_2_0_PCI	**Page Request Control**
00306h	PR_STATUS_0_2_0_PCI	**Page Request Status**
00308h	OPRC_0_2_0_PCI	**Outstanding Page Request Capacity**
0030Ch	OPRA_0_2_0_PCI	**Outstanding Page Request Allocation**
00320h	SRIOV_ECAPHDR_0_2_0_PCI	**SRIOV Extended Capability Header**
00324h	SRIOV_CAP_0_2_0_PCI	**SRIOV Capabilities**

Address	Symbol	Name
00328h	SRIOV_CTRL_0_2_0_PCI	**SRIOV Control Register**
0032Ah	SRIOV_STS_0_2_0_PCI	**SRIOV Status**
0032Ch	SRIOV_INITVFS_0_2_0_PCI	**SRIOV Initial VFs**
0032Eh	SRIOV_TOTVFS_0_2_0_PCI	**SRIOV Total VFs**
00330h	SRIOV_NUMOFVFS_0_2_0_PCI	**SRIOV Number of VFs**
00334h	FIRST_VF_OFFSET_0_2_0_PCI	**SRIOV First VF Offset**
00336h	VF_STRIDE_0_2_0_PCI	**VF Stride**
0033Ah	VF_DEVICEID_0_2_0_PCI	**VF Device ID**
0033Ch	SUPPORTED_PAGE_SIZES_0_2_0_PCI	**Supported Page Sizes**
00340h	SYSTEM_PAGE_SIZES_0_2_0_PCI	**System Page Sizes**
00344h	VF_BAR0_LDW_0_2_0_PCI	**VF BAR0 LDW**
00348h	VF_BAR0_UDW_0_2_0_PCI	**VF BAR0 UDW**
0034Ch	VF_BAR1_LDW_0_2_0_PCI	**VF BAR1 LDW**
00350h	VF_BAR1_UDW_0_2_0_PCI	**VF BAR1 UDW**
00354h	VF_BAR2_LDW_0_2_0_PCI	**VF BAR2 LDW**
00358h	VF_BAR2_UDW_0_2_0_PCI	**VF BAR2 UDW**
0035Ch	VF_MIGST_OFFSET_0_2_0_PCI	**VF Migration State Array Offset**

MMIO

MMIO chapter for System Interfaces Volume.

Force Wake Table

Before initiating reads or writes to power domains that may be off, software must issue the proper "force wake" and verify to guarantee that the register is powered. The following table shows the source and target ranges, and which force wake to use:

Target

<table>
<tr><th>Source</th><th>Uncore</th><th>GTI/Blit</th><th>Render</th><th>Media</th></tr>
<tr><td>CPU</td><td>n/a</td><td>Write A188, then Poll 130044</td><td>Write 0xA278[15:0], then Poll 0xD84[15:0]</td><td>VDBOX0 - Write 0xA540[15:0], Poll 0xD50[15:0] VDBOX1 - Write 0xA544[15:0], Poll 0xD54[15:0]
VDBOX2 - Write 0xA548[15:0], Poll 0xD58[15:0]
VDBOX3 - Write 0xA54C[15:0], Poll 0xD5C[15:0]
VDBOX4 - Write 0xA550[15:0], Poll 0xD60[15:0]
VDBOX5 - Write 0xA554[15:0], Poll 0xD64[15:0]
VDBOX6 - Write 0xA558[15:0], Poll 0xD68[15:0]
VDBOX7 - Write 0xA55C[15:0], Poll 0xD6C[15:0]
VEBOX0 - Write 0xA560[15:0], Poll 0xD70[15:0]
VEBOX1 - Write 0xA564[15:0], Poll 0xD74[15:0]
VEBOX2 - Write 0xA568[15:0], Poll 0xD78[15:0]
VEBOX3 - Write 0xA56C[15:0], Poll 0xD7C[15:0]</td></tr>
<tr><td>CS</td><td>n/a</td><td>n/a</td><td>n/a</td><td>MI_PIPE_SELECT[5]
or
R_PWR_CLK_STATE[19]
(CS/PM handshake internally)</td></tr>
<tr><td>VCS</td><td>n/a</td><td>n/a</td><td>new MI_FORCE_WAKEUP[0]</td><td>n/a</td></tr>
<tr><td>BCS</td><td>n/a</td><td>n/a</td><td>new MI_FORCE_WAKEUP[0]</td><td>new MI_PIPE_SELECT[5]</td></tr>
<tr><td>VECS</td><td>n/a</td><td>n/a</td><td>new MI_FORCE_WAKEUP[0]</td><td>new MI_PIPE_SELECT[5]</td></tr>
<tr><td></td><td></td><td></td><td></td><td>Each media slice is allocated 64KB MMIO space, and each *box gets 16KB range. Here is the list of base address for each VD/VE box.
<table>
<tr><th>Media Boxes</th><th>Base Address</th><th>Offset Range</th><th>Size</th><th>Media sliceid[2:0]</th><th>Media subsliceid[1:0]</th></tr>
<tr><td>VDBOX0</td><td>0x1C_0000</td><td>0x0000 – 0x3FFF</td><td>16KB</td><td>000</td><td>00</td></tr>
<tr><td>VDBOX1</td><td>0x1C_4000</td><td>0x0000 – 0x3FFF</td><td>16KB</td><td>000</td><td>01</td></tr>
<tr><td>VEBOX0</td><td>0x1C_8000</td><td>0x0000 – 0x3FFF</td><td>16KB</td><td>000</td><td>00</td></tr>
</table></td></tr>
</table>

Source	Uncore	GTI/Blit	Render	Media

VDBOX2	0x1D_0000	0x0000 – 0x3FFF	16KB	001	00
VDBOX3	0x1D_4000	0x0000 – 0x3FFF	16KB	001	01
VEBOX1	0x1D_8000	0x0000 – 0x3FFF	16KB	001	00
VDBOX4	0x1E_0000	0x0000 – 0x3FFF	16KB	010	00
VDBOX5	0x1E_4000	0x0000 – 0x3FFF	16KB	010	01
VEBOX2	0x1E_8000	0x0000 – 0x3FFF	16KB	010	00
VDBOX6	0x1F_0000	0x0000 – 0x3FFF	16KB	011	00
VDBOX7	0x1F_4000	0x0000 – 0x3FFF	16KB	011	01
VEBOX3	0x1F_8000	0x0000 – 0x3FFF	16KB	011	00

The VDBOX clients and their offsets within their respective VDBOX range are as follows:

UNIT	Address	Size
VCS (range 0)	0x0000 – 0x07FF	2 KB
MFX (VIN)	0x0800 – 0x0FFF	2 KB
VCS (range 1)	0x1000 – 0x1FFF	4 KB
HuC (HWM/HUWM)	0x2000 – 0x27FF	2 KB
HEVC (HWM)	0x2800 – 0x29FF	512 B
HEVCFE	0x2A00-0x2BFF	512 B
CPCFG	0x3F00-0x3FFF	128B
SCR	n/a	0B (no MMIO, but is a MsgCh EP)

Source	Uncore	GTI/Blit	Render	Media
				Total allocation: 11.125 KB

The VEbox clients and their offsets within their respective VEBOX range are as follows.

UNIT	Address	Size
VECS	0x0000 – 0x1FFF	8 KB
VFW	0x0000 – 0x00FF	256 B
		Total allocation: 8.25 KB

* Note: Some CP registers (0x9400-0x97FF) are replicated in all domains, thus both render and media domains must be awake.

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 inaccessable in a SW-virtualized environment.

Observability

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

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 part of the BSpec 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 pin point 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.

Gen7 enhanced the aggregating counters to support the additional thread types generated by DX11 workloads. The high rate at which interesting internal events can occur motivated adding an interrupt-generation capability so that HW could notify SW when the data buffer was approaching full.

Gen7.5 enhances support for high reporting frequencies by increasing the report buffer size in order to allow SW sufficient time between performance monitoring interrupts, enabling single run histogramming support for events like pixels per polygon. Desire for more flexibility in custom event creation drove addition of 4 more Boolean counters.

Issues with SNB support drove enhancements to enable performance monitoring with RC6 enabled, different report buffer ring wrap behavior, and MMIO visibility into performance counters.

Gen8 enhances functionality of aggregating counters for EUs by providing some flexibility in what quantities are aggregated across all EUs including more quantities relevant to GPGPU workloads. Since several of the previously defined aggregating counters had not delivered very much value on earlier projects, the overall number of A-counters has gone down even though aggregating counters for sampler/pixel-level functionality have been added/redefined. Custom counter creation has been enhanced by adding the ability to negate a signal at the input of the Boolean logic. Given that the increased complexity of GFX workloads and number of EUs in GT2/GT3 could lead to more frequent counter overflows, the width A-counters has increased to 40 bits. HW optimizations have also modified the SW interface slightly. Please note that no media-specific A-counter support was added, hence requiring all media engine support to be implemented using B/C-counters.

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.

CPD must be disabled during performance counting or undercounts may occur.

HW Support

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

Performance Counter Registers

Register
OACTXCONTROL - Observation Architecture Control per Context
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

Register
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

The following Performance Statistics registers are power context save/restored:

Register
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 A2
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

Register
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
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

Register
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
OAPERF_A35 - Aggregate_Perf_Counter_A35
OAPERF_B0 - Boolean_Counter_B0
OAPERF_B1 - Boolean_Counter_B1
OAPERF_B2 - Boolean_Counter_B2
OAPERF_B3 - Boolean_Counter_B3
OAPERF_B4 - Boolean_Counter_B4
OAPERF_B5 - Boolean_Counter_B5
OAPERF_B6 - Boolean_Counter_B6
OAPERF_B7 - Boolean_Counter_B7
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

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 bit could be allocated to some other unit)**

WDBoxOAInterrupt Vector

- IMR

Bit Definition for Interrupt Control Registers

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

OACS 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:

Field	Description
GPU TICKS[31:0]	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.
Context ID[31:0]	This field carries the Context ID of the active context in render/compute engines in the reports generated by context load in case of normal reports or 32bit unique ID provided by SW, in case of MMIO Triggered reports. [31:0]: Context ID in Execlist mode of scheduling. [31:12]: Context ID in Ring Buffer mode of scheduling, [11:0] must be ignored.
Context ID[31:0]	This field carries the Context ID of the active context in render engine. [31:0]: Context ID in Execlist mode of scheduling.
TIME_STAMP[31:0]	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 Vol1C.4 Render Command Streamer BSpec.

SourceID[5:0]

Field	Description
GPU TICKS[31:0]	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.
TIME_STAMP[31:0]	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 Vol1C.4 Render Command Streamer BSpec.

<table>
<tr><th>Field</th><th>Description</th></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>Reserved MBZ</td></tr>
<tr><td>25</td><td>Render Context Valid: When set indicates render context is valid and the ID is of the render context is set in “Context ID” field of report format.</td></tr>
<tr><td>24:19</td><td>Report Reason[5:0]:
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]: Reserved</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 Trigger-1” is given priority over “Internal Report Trigger-2”. By default “Report Trigger Threshold Enable-1” is outputted.</td></tr>
<tr><td>16</td><td>Timer Enabled</td></tr>
<tr><td>15:0</td><td></td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>RPT_ID[31:0]</td><td>Subfields of RPT_ID detailed below:

<table>
<tr><td>31:25</td><td>squashed_slice_clock_frequency[6:0]:

Ratio encoding in this field can be decoded using the ratio encoding table that is part of the definition of the RP_FREQ_NORMAL register.</td></tr>
<tr><td>24:19</td><td>Report Reason[5:0]:
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]: [SKL+]: When set indicates the current report is due to Clock Ratio change between squashed Slice Clock frequency to squashed Unslice clock frequency.</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 Trigger-1” is given priority over “Internal Report Trigger-2”. By default “Report Trigger Threshold Enable-1” is outputted.</td></tr>
<tr><td>16</td><td>Render Context Valid: When set indicates render context is valid and the ID is of the render context is set in “Context ID” field of report format.</td></tr>
<tr><td>15:0</td><td>Additional Report Flags:
When “OA_DEBUG_REGISTER: Disable OA reports due to clock ratio change” is 1, these bits comprise of following:
[10:9]: squashed_slice_clock_frequency [8:7]
[8:0]: squashed_unslice_clock_frequency [8:0]
When “OA_DEBUG_REGISTER:Disable OA reports due to clock ratio change” is 0,
Ratio encoding in this field can be decoded using the ratio encoding table that is part of the definition of the RP_FREQ_NORMAL register.</td></tr>
</table>
</td></tr>
</table>

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 **OACONTROL** register. The organization and number of report formats vary per project and are detailed in Performance Counter Report Formats.

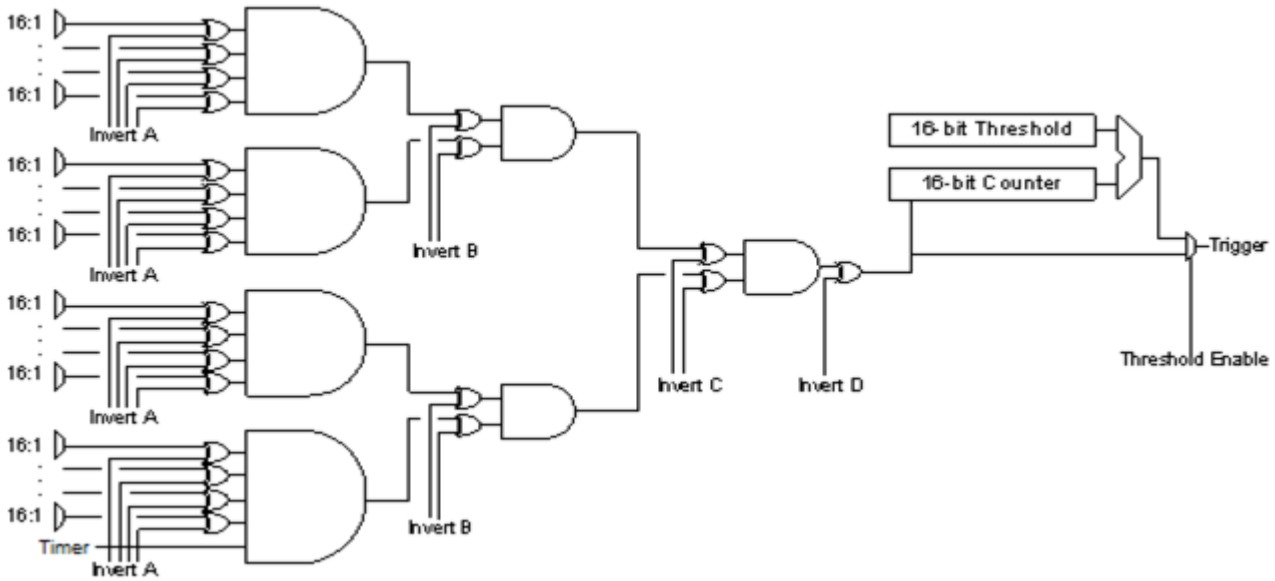
Details of Start Trigger Behavior

- 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 in BDW 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 (**Observation Architecture Control**). 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 (**Observation Architecture Control**). 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: **[Register] Aggregate Perf Counter A0**
A1	# of Vertex Shader Threads Dispatched	Count of VS threads dispatched to EUs Link to detailed register definition: **[Register] Aggregate Perf Counter A1**
A2	# of Hull Shader Threads Dispatched	Count of HS threads dispatched to EUs Link to detailed register definition: **[Register] Aggregate Perf Counter A2**

Counter #	Event	Description
A3	# of Domain Shader Threads Dispatched	Count of DS threads dispatched to EUs Link to detailed register definition: **[Register] Aggregate Perf Counter A3**
A4	# of GPGPU Threads Dispatched	Count of GPGPU threads dispatched to EUs Available on both qualified and free-running counters Link to detailed register definition: **[Register] Aggregate Perf Counter A4**
A5	# of Geometry Shader Threads Dispatched	Count of GS threads dispatched to EUs Link to detailed register definition: **[Register] Aggregate Perf Counter A5**
A6	# of Pixel Shader Threads Dispatched	Count of PS threads dispatched to EUs Available on both qualified and free-running counters Link to detailed register definition: **[Register] Aggregate Perf Counter A6**
A7	Aggregating EU counter 0	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition: **[Register] Aggregate Perf Counter A7**
A8	Aggregating EU counter 1	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition: **[Register] Aggregate Perf Counter A8**
A9	Aggregating EU counter 2	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition: **[Register] Aggregate Perf Counter A9**
A10	Aggregating EU counter 3	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition: **[Register] Aggregate Perf Counter A10**
A11	Aggregating EU counter 4	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition: **[Register] Aggregate Perf Counter A11**

Counter #	Event	Description
A12	Aggregating EU counter 5	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition: **[Register] Aggregate Perf Counter A12**
A13	Aggregating EU counter 6	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition: **[Register] Aggregate Perf Counter A13**
A14	Aggregating EU counter 7	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition: **[Register] Aggregate Perf Counter A14**
A15	Aggregating EU counter 8	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition: **[Register] Aggregate Perf Counter A15**
A16	Aggregating EU counter 9	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition: **[Register] Aggregate Perf Counter A16**
A17	Aggregating EU counter 10	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition: **[Register] Aggregate Perf Counter A17**
A18	Aggregating EU counter 11	User-defined (details in Flexible EU Event Counters section) Link to detailed register definition: **[Register] Aggregate Perf Counter A18**
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: **[Register] Aggregate Perf Counter A19**
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: **[Register] Aggregate Perf Counter A20**

Counter #	Event	Description
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.) Link to detailed register definition: **[Register] Aggregate Perf Counter A21**
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: **[Register] Aggregate Perf Counter A22**
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: **[Register] Aggregate Perf Counter A23**
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: **[Register] Aggregate Perf Counter A24**
A25	2x2s Failing post-PS Tests	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: **[Register] Aggregate Perf Counter A25**

Counter #	Event	Description
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: **[Register] Aggregate Perf Counter A26**
A27	Blended 2x2s Written to Render Target	Number of samples of blendable 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: **[Register] Aggregate Perf Counter A27**
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: **[Register] Aggregate Perf Counter A28**
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: **[Register] Aggregate Perf Counter A29**

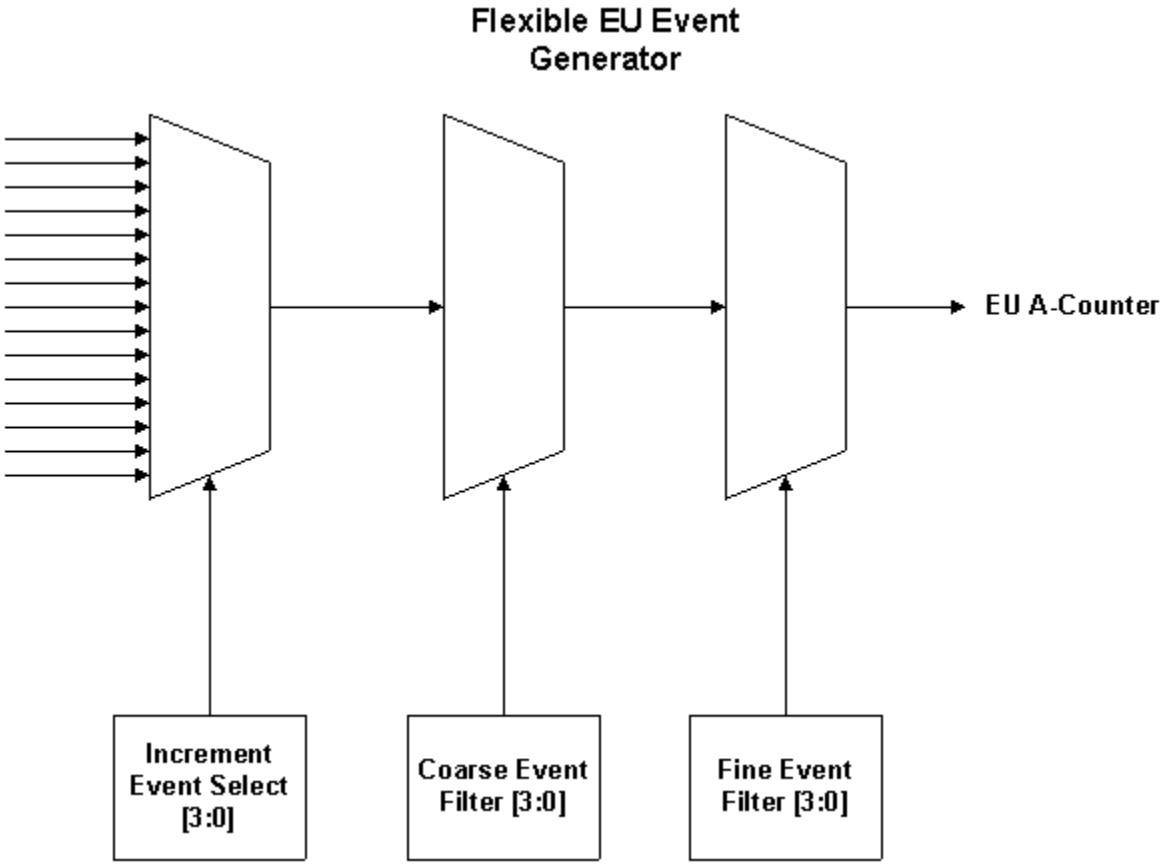
Counter #	Event	Description
A30	SLM Reads	Total read requests from an EU to SLM (including reads generated by atomic operations). Link to detailed register definition: **[Register] Aggregate Perf Counter A30**
A31	SLM Writes	Total write requests from an EU to SLM (including writes generated by atomic operations). Link to detailed register definition: **[Register] Aggregate_Perf_Counter_A31**
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: **[Register] Aggregate_Perf_Counter_A32**
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: **[Register] Aggregate_Perf_Counter_A33**
A34	Atomic Accesses	Aggregated total atomic accesses from all EUs. This counter increments on atomic accesses to both SLM and URB. Link to detailed register definition: **[Register] Aggregate_Perf_Counter_A34** **Gen11 bug and Workaround** SLM atomics are not included in Gen11 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: **[Register] Aggregate_Perf_Counter_A35**

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), BDW 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 FPU0 Pipeline Active	0b0000	Signal that is high on every EU clock where the EU FPU0 pipeline is actively executing a Gen ISA instruction.
EU FPU1 Pipeline Active	0b0001	Signal that is high on every EU clock where the EU FPU1 pipeline is actively executing a Gen ISA instruction.
EU SEND Pipeline Active	0b0010	Signal that is high on every EU clock where the EU send pipeline is actively executing a Gen ISA instruction. Updated event to include new systolic pipeline addition on EU from Gen12 HP onwards. Only fine event filters 0b0000,0b0101, 0b0111, 0b1000, 0b1001, and 0b1010 are supported with this increment event.
EU FPU0 & FPU1 Pipelines Concurrently Active	0b0011	Signal that is high on every EU clock where the EU FPU0 and FPU1 pipelines are both actively executing a Gen 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.

Increment Event	Encoding	Notes
Some EU Pipeline Active	0b0100	Signal that is high on every EU clock where at least one EU pipeline is actively executing a Gen ISA instruction. Updated event to include new systolic pipeline addition on EU from Gen12 HP onwards. 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.
At Least 1 Thread Loaded But No EU Pipeline Active	0b0101	Signal that is high on every EU clock where at least one thread is loaded but no EU pipeline is actively executing a Gen ISA instruction. Updated event to include new systolic pipeline addition on EU from Gen12 HP onwards. 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.
Threads loaded integrator == max threads for current HW SKU	0b1000	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. **Note:** There were no C steppings for the BDW GPU, so this increment event is supported for BDW:*:E0 and later. **Note:** A thread slot is considered to be "loaded" once the EU receives the transparent header. Hence, time spent transferring additional thread header data phases (e.g. push constants and vertex attribute data) will count towards thread occupancy.
No event	0b1111	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 0b0000/0b0001/0b0010, masks increment events unless the FFID which dispatched the currently executing thread equals FFID of VS.
HS Thread Filter	0b0010	For increment events 0b0000/0b0001/0b0010, masks increment event unless the FFID which dispatched the currently executing thread equals FFID of HS.
DS Thread Filter	0b0011	For increment events 0b0000/0b0001/0b0010, masks increment event unless the FFID which dispatched the currently executing thread equals FFID of DS.

Coarse Event Filter	Encoding	Notes
GS Thread Filter	0b0100	For increment events 0b0000/0b0001/0b0010, masks increment event unless the FFID which dispatched the currently executing thread equals FFID of GS.
PS Thread Filter	0b0101	For increment events 0b0000/0b0001/0b0010, masks increment event unless the FFID which dispatched the currently executing thread equals FFID of PS.
TS Thread Filter	0b0110	For increment events 0b0000/0b0001/0b0010, masks increment event unless the FFID which dispatched the currently executing thread equals FFID of 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).
Row = 1	0b1000	Masks increment event unless the row ID for this EU is 1 (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.

Fine Event Filter	Encoding	Notes
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

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.