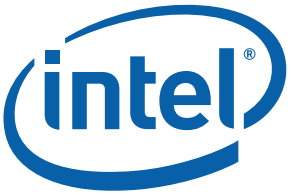


Intel® Xeon® Processor D-1500 Product Family Uncore Performance Monitoring Reference Manual

Revision 1.0

May 2015

Document Number: 332427-001

Contents

Figures

Tables

Revision History

DocID	Revision	Description	Date
332427-001	1.0	• Initial release	May 2015

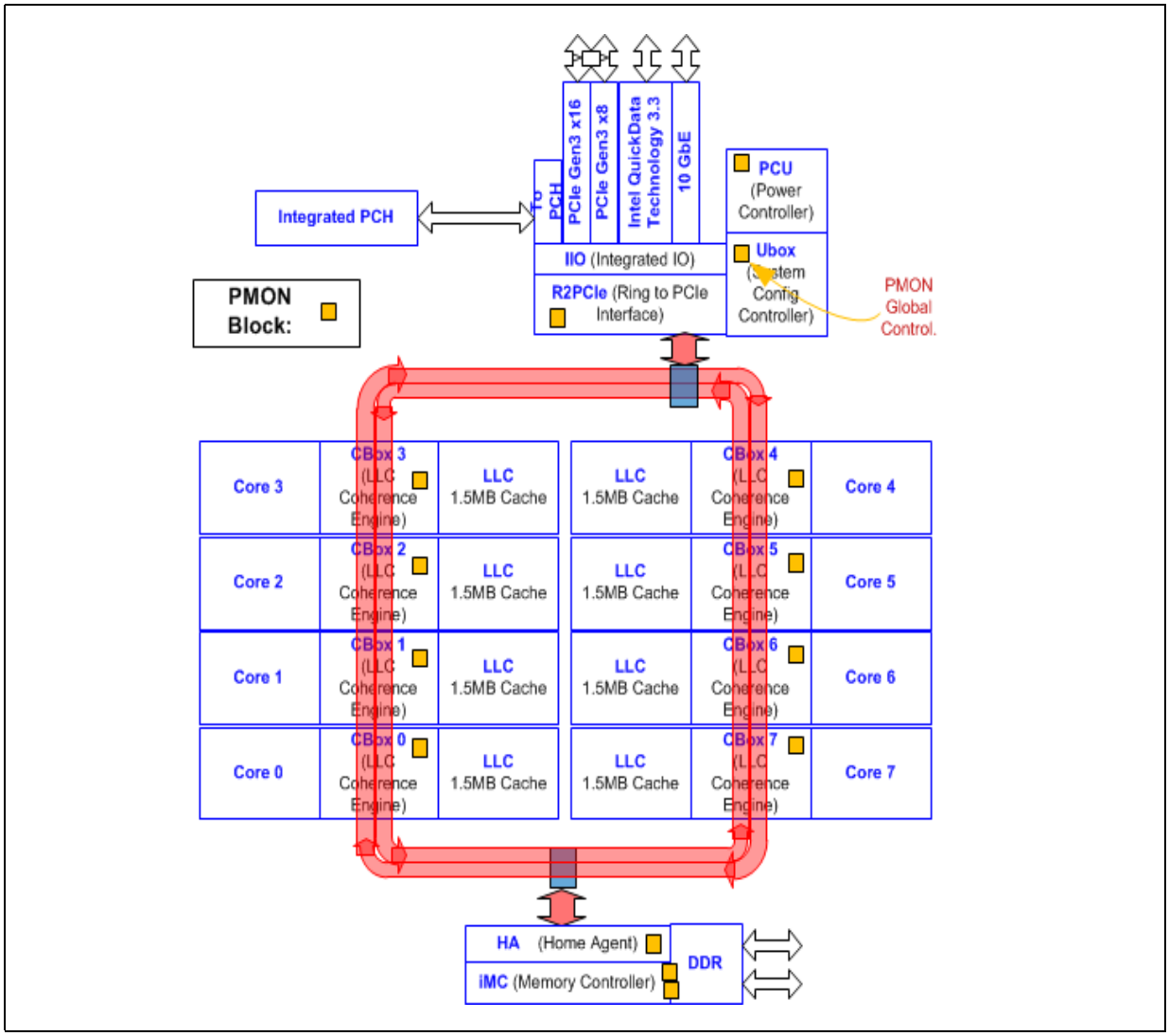
§

1 Introduction

1.1 Introduction

The uncore sub-system of the next generation Intel® Xeon® Processor D-1500 Product Family is shown in Figure 1-1. The uncore sub-system consists of a variety of components, ranging from the CBox caching agent to the power controller unit (PCU), integrated memory controller (IMC) and home agent (HA), to name a few. Most of these components provide similar performance monitoring capabilities.

Figure 1-1. Intel® Xeon® Processor D-1500 Product Family -8C Block Diagram



Note: This diagram represents one possible configuration. The number of supported cores vary by SKU. Not all features supported on all SKUs.

1.2 Uncore PMON Overview

The uncore performance monitoring facilities are organized into per-component performance monitoring (or '**PMON**') units. A PMON unit within an uncore component may contain one or more sets of counter registers. With the exception of the UBox, each PMON unit provides a unit-level control register to synchronize actions across the counters within the box (e.g. to start/stop counting).

Events can be collected by reading a set of local counter registers. Each counter register is paired with a dedicated control register used to specify what to count (i.e. through the event select/umask fields) and how to count it. Some units provide the ability to specify additional information that can be used to 'filter' the monitored events (e.g., CBox; see Section 2.3.2.3, "CBo Filter Registers (Cn_MSR_PMON_BOX_FILTER{0,1})").

Each of these boxes communicates with the U-Box which contains registers to control all uncore PMU activity (as outlined in Section 2.1, "Uncore Per-Socket Performance Monitoring Control").

Uncore performance monitors represent a per-socket resource that is not meant to be affected by context switches and thread migration performed by the OS, it is recommended that the monitoring software agent establish a fixed affinity binding to prevent cross-talk of event counts from different uncore PMU.

The programming interface of the counter registers and control registers fall into two address spaces:

- Accessed by MSR are PMON registers within the Cbo units, PCU, and U-Box, see Table 1-2.
- Access by PCI device configuration space are PMON registers within the HA, IMC and R2PCIe units, see Table 1-3.

Irrespective of the address-space difference and with only minor exceptions, the bit-granular layout of the control registers to program event code, unit mask, start/stop, and signal filtering via threshold/edge detect are the same.

Software may be notified of an overflowing uncore counter on any core.

The general performance monitoring capabilities of each box are outlined in the following table.

Table 1-1. Per-Box Performance Monitoring Capabilities

Box	# Boxes	# Counters/Box	# Queue Enabled	Packet Match/Mask Filters?	Bit Width
CBox	up to 8	4	1	Y	48
HA	1	4	4	Y	48
IMC	1	4 (+1) (per channel)	4	N	48
PCU	1	4 (+2)	4	N	48
R2PCIe	1	4	1	N	48
UBox	1	2 (+1)	0	N	48
IRP	1	4	4	N	48

1.3 Section References

The following sections provide a breakdown of the performance monitoring capabilities for each box.

- Section 2.1, "Uncore Per-Socket Performance Monitoring Control"
- Section 2.2, "UBox Performance Monitoring"

- Section 2.3, “Caching Agent (Cbo) Performance Monitoring”
- Section 2.4, “Home Agent (HA) Performance Monitoring”
- Section 2.5, “Memory Controller (IMC) Performance Monitoring”
- Section 2.6, “IRP Performance Monitoring”
- Section 2.7, “Power Control (PCU) Performance Monitoring”
- Section 2.8, “R2PCIe Performance Monitoring”

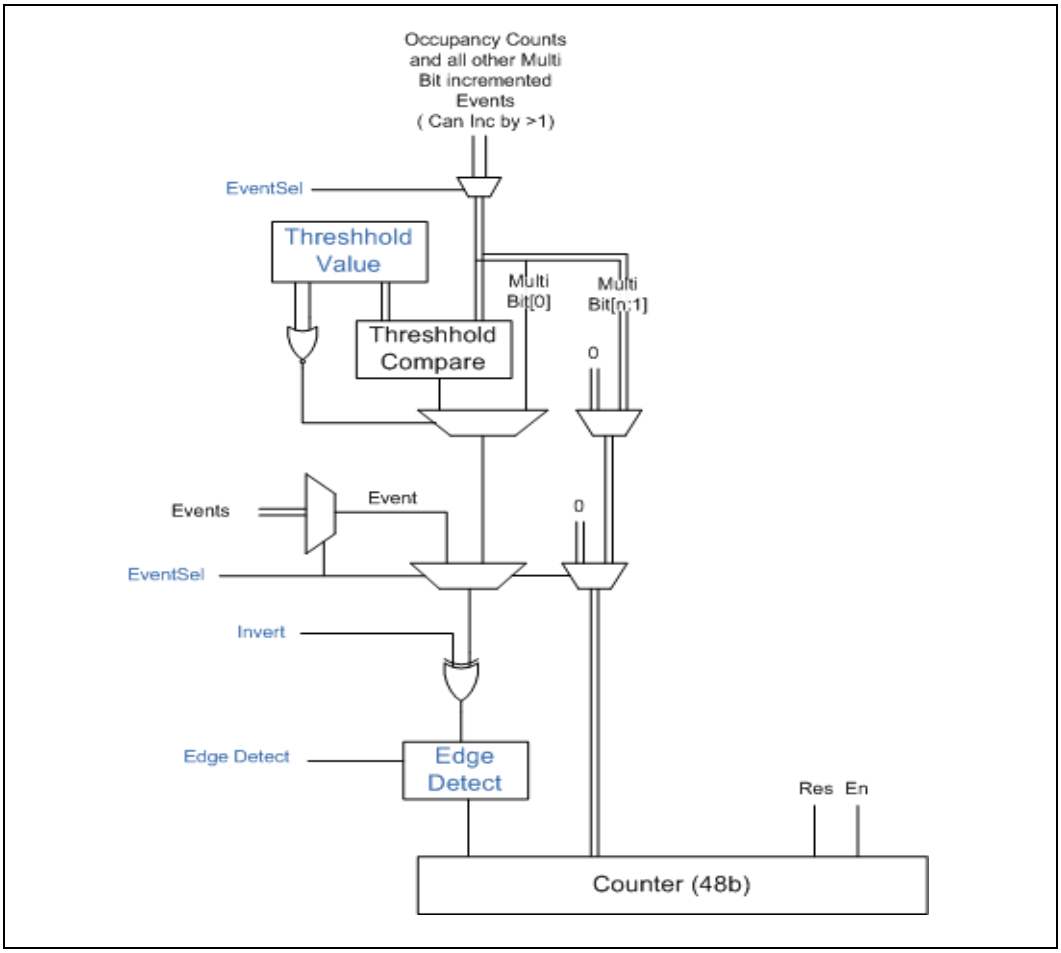
1.4 Uncore PMON - Typical Counter Control Logic

Following is a diagram of the standard perfmon counter control block illustrating how event information is routed, selected, filtered (by other bits in the control register) and sent to the paired data register for storage.

Details for how control bits affect event information is presented in each of the box subsections of Chapter 2, with some summary information below.

Note: The PCU uses an adaptation of this block (refer to Section 2.7.2, “PCU Performance Monitors” more information). Also note that only a subset of the available control bits are presented in the diagram.

Figure 1-2. Perfmon Counter Control Block Diagram



Selecting What To Monitor: The main task of a configuration register is to select the event to be monitored by its respective data counter. Setting the *.ev_sel* and *.umask* fields performs the event selection.

Note: Only the *.ev_sel* is pictured in the previous figure. The *.umask* field is generally used to select subevents of the event. Once the proper subevent combination has been selected, it is passed on to the per Counter EventSel Mux.

Additional control bits used to filter and create information related to the selected Event:

Applying a Threshold to Incoming Events: *.thresh* - since most counters can increment by a value greater than 1, a threshold can be applied to generate an event based on the outcome of the comparison. If *.thresh* is set to a non-zero value, that value is compared against the incoming count for that event in each cycle. If the incoming count is >= the threshold value, then the event count captured in the data register will be incremented by 1.

Using the threshold field to generate additional events can be particularly useful when applied to a queue occupancy count. For example, if a queue is known to contain eight entries, it may be useful to know how often it contains 6 or more entries (i.e. Almost Full) or when it contains 1 or more entries (i.e. Not Empty).

Note: For Intel® Xeon® Processor D-1500 Product Family the *.invert* and *.edge_det* bits follow the threshold comparison in sequence. If a user wishes to apply these bits to events that only increment by 1 per cycle, *thresh* must be set to 0x1.

Inverting the Threshold Comparison: *.invert* - Changes *.thresh* test condition to '<'.

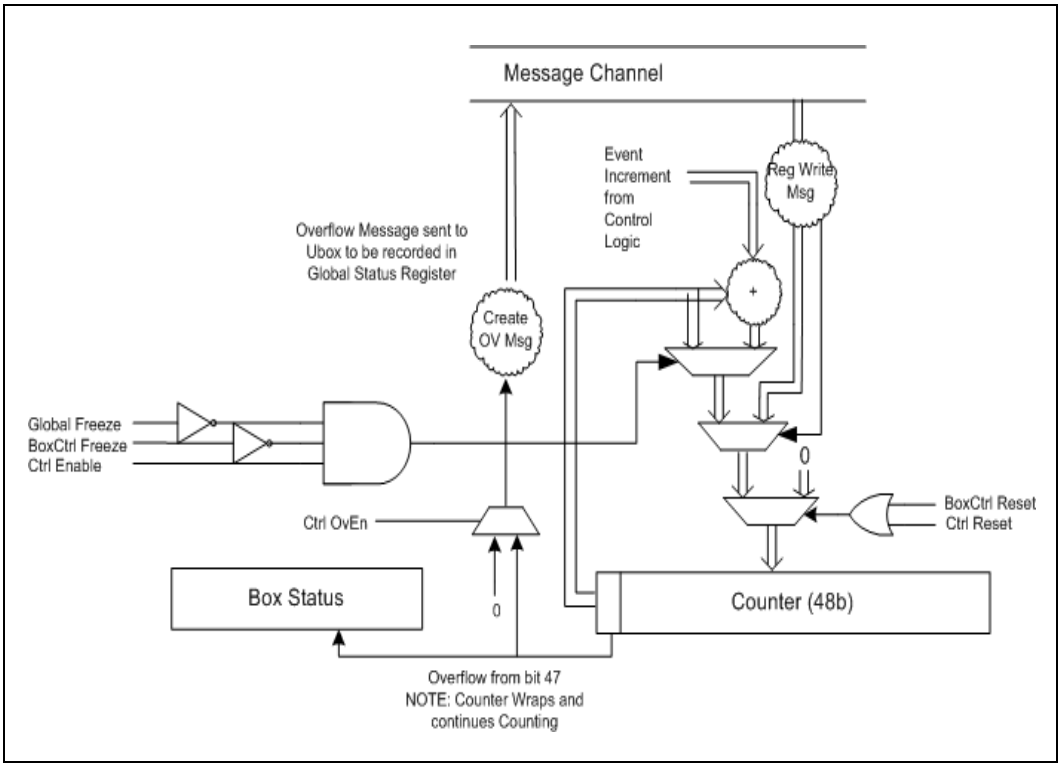
Counting State Transitions Instead of per-Cycle Events: *.edge_det* - Rather than accumulating the raw count each cycle (for events that can increment by 1 per cycle), the register can capture transitions from no event to an event incoming (i.e. the 'Rising Edge').

1.5 Uncore PMON - Typical Counter Logic

Following is a diagram of the standard perfmon counter block illustrating the control for managing the Data Register including how to start/stop the register, reset it, indicate an overflow and capture the information sent from the Counter Control block. The diagram contains bits from all level in the Counter Control Hierarchy - Global (found in the UBox), Box level as well as the Counter's Control register.

Details on how to perform counter management, including how to set up a Monitoring Session and periodically sample the counters can be found in Chapter 2.

Figure 1-3. Perfmon Counter Block Diagram



Telling HW that the Control Register Is Set: *.en* bit must be set to 1 to enable counting. Once counting has been enabled at all levels of the Performance Monitoring Hierarchy (refer to Section 2.1.2, "Setting up a Monitoring Session" for more information), the paired data register will begin to collect events.

Notification after X events: *.ov_en* - Instead of manually stopping the counters at intervals (often wall clock time) pre-determined by software, hardware can be set to notify monitoring software when a set number of events has occurred. The Overflow Enable bit is provided for just that purpose. See Section 2.1.1, "Counter Overflow" for more information on how to use this mechanism.

1.6 Uncore PMU Summary Tables

Following is the list of registers provided in the Intel® Xeon® Processor D-1500 Product Family Uncore for Performance Monitoring. Performance monitors are split between MSR space (U, CBo and PCU) and PCICFG space.

Note: The number of CBoxes varies with the number of Cores in a system. To determine the number of CBoxes, SW should read bits 15:0 in the CAPID5 register located at Device 30, Function 3, Offset 0x98. These 16 bits form a bit vector of available LLC slices and the CBoxes that manage those slices. For example: If bits 15:0 read 0x0F0F, the PMON blocks corresponding to CBoxes 0-3 and 8-11 are available and CBoxes 4-7 and 12-15 are not available.

Table 1-2. MSR Space UncorePerformance Monitoring Registers (Sheet 1 of 2)

Box	MSR Addresses	Description
CBox Counters		
CBox 15	0xEFB-0xEF8	Counter Registers
	0xEF5,0xEF6	Counter Filters
	0xEF4-0xEF1	Counter Config Registers
	0xEF0,0xEF7	Box Control/Status
CBox 14	0xEEB-0xEE8	Counter Registers
	0xEE5,0xEE6	Counter Filters
	0xEE4-0xEE1	Counter Config Registers
	0xEE0,0xEE7	Box Control/Status
CBox 13	0xEDB-0xED8	Counter Registers
	0xED5,0xED6	Counter Filters
	0xED4-0xED1	Counter Config Registers
	0xED0,0xED7	Box Control/Status
CBox 12	0xECB-0xEC8	Counter Registers
	0xEC5,0xEC6	Counter Filters
	0xEC4-0xEC1	Counter Config Registers
	0xEC0,0xEC7	Box Control/Status
CBox 11	0xEBB-0xEB8	Counter Registers
	0xEB5,0xEB6	Counter Filters
	0xEB4-0xEB1	Counter Config Registers
	0xEB0,0xEB7	Box Control/Status
CBox 10	0xEAB-0xEA8	Counter Registers
	0xEA5,0xEA6	Counter Filters
	0xEA4-0xEA1	Counter Config Registers
	0xEA0,0xEA7	Box Control/Status
CBox 9	0xE9B-0xE98	Counter Registers
	0xE95,0xE96	Counter Filters
	0xE94-0xE91	Counter Config Registers
	0xE90,0xE97	Box Control/Status
CBox 8	0xE8B-0xE88	Counter Registers
	0xE85,0xE86	Counter Filters
	0xE84-0xE81	Counter Config Registers
	0xE80,0xE87	Box Control/Status
CBox 7	0xE7B-0xE78	Counter Registers
	0xE75,0xE76	Counter Filters
	0xE74-0xE71	Counter Config Registers
	0xE70,0xE77	Box Control/Status
CBox 6	0xE6B-0xE68	Counter Registers
	0xE65,0xE66	Counter Filters
	0xE64-0xE61	Counter Config Registers
	0xE60,0xE67	Box Control/Status

Table 1-2. MSR Space UncorePerformance Monitoring Registers (Sheet 2 of 2)

Box	MSR Addresses	Description
CBox 5	0xE5B-0xE58	Counter Registers
	0xE55,0xE56	Counter Filters
	0xE54-0xE51	Counter Config Registers
	0xE50,0xE57	Box Control/Status
CBox 4	0xE4B-0xE48	Counter Registers
	0xE45,0xE46	Counter Filters
	0xE44-0xE41	Counter Config Registers
	0xE40,0xE47	Box Control/Status
CBox 3	0xE3B-0xE38	Counter Registers
	0xE35,0xE36	Counter Filters
	0xE34-0xE31	Counter Config Registers
	0xE30,0xE37	Box Control/Status
CBox 2	0xE2B-0xE28	Counter Registers
	0xE25,0xE26	Counter Filters
	0xE24-0xE21	Counter Config Registers
	0xE20,0xE27	Box Control/Status
CBox 1	0xE1B-0xE18	Counter Registers
	0xE15,0xE16	Counter Filters
	0xE14-0xE11	Counter Config Registers
	0xE10,0xE17	Box Control/Status
CBox 0	0xE0B-0xE08	Counter Registers
	0xE05,0xE06	Counter Filters
	0xE04-0xE01	Counter Config Registers
	0xE00,0xE07	Box Control/Status
PCU Counters		
	0x71A-0x717	Counter Registers
	0x715	Counter Filters
	0x714-0x711	Counter Config Registers
	0x710,0x716	Box Control/Status
	0x3FD-0x3FC	Fixed Counters (Non-PMON)
U-Box Counters		
For U-Box	0x70A-0x709	Counter Registers
	0x708	Box Status
	0x706-0x705	Counter Config Registers
	0x704,0x703	Fixed Counter/Config Register
U-Box Counters		
For Global Control		
	0x700,0x701	Global Control/Status

Table 1-3. PCICFG Space Uncore Performance Monitoring Registers

Box	PCICFG Register Addresses	Device ID	Description
HA0	**D18:F1**	**0x6F30**	
	F8-F4		Box Control/Status
	E4-D8		Counter Config Registers
	BC-A0		Counter Registers
	48-40		Opcode/Addr Match Filters
iMC0	**D20:F0,1**	**0x6FB4, 0x6FB5**	**D20:F0,F1 For Channel 0,1**
	F8-F4		Box Control/Status
	F0		Counter Config Register (Fixed)
	E4-D8		Counter Config Registers (General)
	D4-D0		Counter Register (Fixed)
	BC-A0		Counter Registers (General)
IRP	**D5:F6**	**0x6F39**	
	F8-F4		Box Control/Status
	E4-E0 & DC-D8		Counter Config Registers
	C0-B8 & B0-A0		Counter Registers
R2PCIe	**D16:F1**	**0x6F34**	
	F8-F4		Box Control/Status
	E4-D8		Counter Config Registers
	BC-A0		Counter Registers

1.6.1 On Finding the Package's Bus number for Uncore PMON registers in PCICfg Space

PCI-based uncore units can be found using bus, device and functions numbers. However, the **busno** has to be found dynamically in each package. The code is embedded below.

First, for each package, it is necessary to read the node ID offset in the Ubox. That needs to match the GID offset of the Ubox in a specific pattern to get the busno for the package. This busno can then be used with the given D:F (device:function) listed with each box's counters that are accessed through PCICfg space (Table 1-3, "PCICFG Space Uncore Performance Monitoring Registers," on page 16).

```
#define BROADWELL_SERVER_SOCKETID_UBOX_DID      0x6F1e

//the below LNID and GID apply for SNB-EP, IVB and Broadwell uServer
#define UNC_SOCKETID_UBOX_LNID_OFFSET      0x40
#define UNC_SOCKETID_UBOX_GID_OFFSET       0x54

for (bus_no = 0; bus_no < 256; bus_no++) {
   for (device_no = 0; device_no < 32; device_no++) {
      for (function_no = 0; function_no < 8; function_no++) {

         // find bus, device, and function number for socket ID UBOX device
         pci_address = FORM_PCI_ADDR(bus_no, device_no, function_no, 0);
```

```
value = PCI_Read_Ulong(pci_address);

vendor_id = value & VENDOR_ID_MASK;
device_id = (value & DEVICE_ID_MASK) >> DEVICE_ID_BITSHIFT;

if (vendor_id != DRV_IS_PCI_VENDOR_ID_INTEL) {
    continue;
       }
if (device_id == socketid_ubox_did) {
   // first get node id for the local socket
   pci_address = FORM_PCI_ADDR(bus_no, device_no, function_no,
                               UNC_SOCKETID_UBOX_LNID_OFFSET);
   gid = PCI_Read_Ulong(pci_address) & 0x00000007;

   // Get the node id mapping register:
   // Basic idea is to read the Node ID Mapping Register (below)
   // and match one of the nodes with gid that we read above
   // from the Node ID configuration register (above).
   // Every three bits in the Node ID Mapping Register maps to a
   // particular node (or package). Bits 2:0 maps to package 0,
   // bits 5:3 maps to package 1, and so on. Thus, we have to
   // parse every triplet of bits to find the match.

   pci_address = FORM_PCI_ADDR(bus_no, device_no, function_no,
                               UNC_SOCKETID_UBOX_GID_OFFSET);
   mapping = PCI_Read_Ulong(pci_address);

   // bits 2:0
   if ((mapping & 0x00000007) == gid) {
      gid = 0;
   }
   // bits 5:3
   else if ((mapping & 0x00000038) == gid) {
       gid = 1;
   }
   // bits 8:6
   else if ((mapping & 0x000001C0) == gid) {
       gid = 2;
   }
   // bits 11:9
   else if ((mapping & 0x00000E00) == gid) {
       gid = 3;
   }
   // bits 14:12
   else if ((mapping & 0x00007000) == gid) {
       gid = 4;
   }
   // bits 17:15
   else if ((mapping & 0x00038000) == gid) {
       gid = 5;
   }
```

```
            // bits 20:18
            else if ((mapping & 0x001C0000) == gid) {
                gid = 6;
            }
            // bits 23:21
            else if ((mapping & 0x00700000) == gid) {
                gid = 7;
            }
            UNC_UBOX_package_to_bus_map[gid] = bus_no;
        }
    }
  }
}
```

1.7 On Parsing and Using Derived Events

For many of the sections covering each box's Performance Monitoring capabilities, a set of commonly measured metrics (or 'Derived Events') has been included. For the most part, these derived events are simple mathematical combinations of events found within the box. However, there are some extensions to the notation used by the metrics.

The following is a breakdown of a CBox Derived Event to illustrate a couple of the notations used.

To calculate "Average Number of Data Read Entries that Miss the LLC when the TOR is not empty".

(TOR_OCCUPANCY.MISS_OPCODE / COUNTER0_OCCUPANCY{edge_det,thresh=0x1}))
with:Cn_MSR_PMON_BOX_FILTER1.opc=0x182.

First term is a normal Event/Subevent.

Second Term requires setting extra control bits in the register the event has been programmed in:

- event_name[.subevent_name]{ctrl_bit[=value],}
- e.g. `COUNTER0_OCCUPANCY{edge_det,thresh=0x1}`

NOTE: If there is no [=value] specified it is assumed that the bit must be set to 1.

Third Term requires programming an extra control register (often for filtering):

- For a single field: with:`Register_Name.field=value1`
- For multiple fields: with:`Register_Name.{field1,field2,...}={value1,value2,...}`
- e.g. `with:Cn_MSR_PMON_BOX_FILTER1.{opc,nid}={0x182,my_node}`

Following is a breakdown of an IMC Derived Event to illustrate a couple more of the notations used.

To calculate "Percent Cycles DRAM Rank x in CKE".

POWER_CKE_CYCLES.RANKx / MC_Chy_PCI_PMON_CTR_FIXED

First Term requires more input to software to determine the specific event/subevent

- In some cases, there may be multiple events/subevents that cover the same information across multiple like hardware units. Rather than manufacturing a derived event for each combination, the derived event will use a lower case variable in the event name.

- e.g. `POWER_CKE_CYCLES.RANKx` where 'x' is a variable to cover events `POWER_CKE_CYCLES.RANK0` through `POWER_CKE_CYCLES.RANK7`

Second Term requires reading a fixed data register

- For the case where the metric requires the information contained in a fixed data register, the pnemonic for the register will be included in the equation. Software will be responsible for configuring the data register and setting it to start counting with the other events used by the metric.
- e.g. `MC_Chy_PCI_PMON_CTR_FIXED`

In addition to these formats, some equations require gathering of extra information outside the box (often for common terms):

- See following section for a breakdown of common terms found in Derived Events.

1.7.1 On Common Terms found in Derived Events

To convert a Latency term from a count of clocks to a count of nanoseconds:

- (**Latency Metric**) - {Box}_CLOCKTICKS * (1000 / UNCORE_FREQUENCY)

To convert a Bandwidth term from a count of raw bytes at the operating clock to GB/sec:

- ((**Traffic Metric in Bytes**) / (TOTAL_INTERVAL / (TSC_SPEED * 1000000))) / GB_CONVERSION
- e.g. For READ_MEM_BW, an event derived from IMC:CAS_COUNT.RD * 64, which is the amount of memory bandwidth consumed by read requests, put 'READ_MEM_BW' into the bandwidth term to convert the measurement from raw bytes to GB/sec.

Following are some other terms that may be found within Metrics and how they should be interpreted.

- GB_CONVERSION: 1024^3
- TSC_SPEED: Time Stamp Counter frequency in MHz
- TOTAL_INTERVAL: Overall sample interval (TSC) for the instructions retired event. Typically used to compute a per send metric. Dividing the TOTAL_INTERVAL by CPU_SPEED * 1,000,000 is the number of seconds in the sample interval.
- TOTAL_PROC_CYC: Total number of CPU cycles for a processor event value. Used with processor event data to determine time or work per time as in MB/sec.
- IMC_CHANNELS: Up to 2 for Intel® Xeon® Processor D-1500 Product Family

§

2 Intel® Xeon® Processor D-1500 Product Family Uncore Performance Monitoring

2.1 Uncore Per-Socket Performance Monitoring Control

To manage the large number of counter registers distributed across many units and collect event data efficiently, this section describes the hierarchical technique to start/stop/restart event counting that a software agent may need to perform during a monitoring session.

2.1.1 Counter Overflow

If a box's counter overflows, it can send an overflow message to a global PMON manager (the UBox). To do so, the *.ov_en* bit in the counter's control register must be set to 1. The overflow will then be picked up and the box sending the overflow will be recorded in the UBox.

Each box in the Intel® Xeon® Processor D-1500 Product Family uncore with performance monitors may be configured to respond to this overflow with two basic actions:

2.1.1.1 Freezing on Counter Overflow

Upon receipt of an overflow message from any box, the UBox will assert the global freeze signal. Once the global freeze has been detected, each box will disable (or 'freeze') all of its counters.

2.1.1.2 PMI on Counter Overflow

Upon receipt of the overflow message, the UBox can send a PMI signal to the core executing the monitoring software. To do so, the U_MSR_PMON_GLOBAL_CTL.pmi_core_sel file must be set to point to the core the monitoring software is executing on.

2.1.2 Setting up a Monitoring Session

On HW reset, all the counters are disabled. Enabling is hierarchical. So the following steps, which include programming the event control registers and enabling the counters to begin collecting events, must be taken to set up a monitoring session. Section 2.1.3 covers the steps to stop/re-start counter registers during a monitoring session.

Global Settings in the UBox: (NOTE: Necessary for U-Box monitoring).

a) Freeze all the uncore counters by setting U_MSR_PMON_GLOBAL_CTL.frz_all to 1

OR (if box level freeze control preferred)

a) Freeze the box's counters while setting up the monitoring session.

   e.g., set Cn_MSR_PMON_BOX_CTL.frz to 1

For each event to be measured within each box:

b) Enable counting for each monitor

e.g. Set C0_MSR_PMON_CTL2.en to 1

Note: Recommended: set the *.en* bit for all counters in each box a user intends to monitor, and left alone for the duration of the monitoring session.

Note: For cases where there is no sharing of these counters among software agents independently sampling the counters, software could set the enable bits for all counters it intends to use during the setup phase. For cases where sharing is expected, each agent could use the individual enable bits in order to perform sampling rather than using the box-level freeze from steps (a) and (d).

c) Select event to monitor if the event control register hasn't been programmed:

Program the *.ev_sel* and *.umask* bits in the control register with the encoding necessary to capture the requested event along with any signal conditioning bits (*.thresh*/*.edge_det*) used to qualify the event.

e.g. Set C0_MSR_PMON_CT2.{ev_sel, umask} to {0x03, 0x1} in order to capture LLC_VICTIMS.M_STATE in CBo 0's C0_MSR_PMON_CTR2.

Note: It is also important to program any additional filter registers used to further qualify the events (e.g. setting the opcode match field in Cn_MSR_BOX_FILTER1 to qualify TOR_INSERTS by a specific opcode).

Back to the box level:

d) Reset counters in each box to ensure no stale values have been acquired from previous sessions. Resetting the control registers, particularly those that won't be used is also recommended if for no other reason than to prevent errant overflows. To reset both the counters and control registers write the following registers:

- For each CBox, set Cn_MSR_PMON_BOX_CTL[1:0] to 0x3.
- For each HA, set HAn_PCI_PMON_BOX_CTL[1:0] to 0x3.
- For each DRAM Channel, set MCn_CHy_PCI_PMON_BOX_CTL[1:0] to 0x3.
- Set PCU_MSR_PMON_BOX_CTL[1:0] to 0x3.
- Set R2_PCI_PMON_BOX_CTL[1:0] to 0x3.

Note: The UBox does not have a Box Control register. The counters will need to be manually reset by writing a 0 in each data register.

e) Select how to gather data. *If polling, skip to f.* If sampling:

To set up a **sample interval**, software can pre-program the data register with a value of [2^(register bit width - up to 48) - sample interval length]. Doing so allows software, through use of the pmi mechanism, to be **notified** when the number of events in the sample have been captured. Capturing a performance monitoring sample every 'X cycles' (the fixed counter in the UBox counts uncore clock cycles) is a common use of this mechanism.

i.e. To stop counting and receive notification when the 1,000,000th idle flit is transmitted from Intel QPI on Port 0

- set Q_P0_PCI_PMON_CTR1 to (2^48- 1000)
- set Q_P0_PCI_PMON_CTL1.ev_sel to 0x0
- set Q_P0_PCI_PMON_CTL1.umask to 0x1
- set U_MSR_PMON_GLOBAL_CTL.pmi_core_sel to which core the monitoring thread is executing on.

f) Enable counting at the global level by setting the U_MSR_PMON_GLOBAL_CTL.unfrz_all bit to 1.

OR

f) Enable counting at the box level by unfreezing the counters in each box

e.g. set Cn_MSR_PMON_BOX_CTL.frz to 0

And with that, counting will begin.

Note: The UBox does not have a Box Control register, so there's no box-level freeze to help isolate the UBox from agents counting in other boxes. Once enabled and programmed with a valid event, the UBox counters will collect events. For somewhat better synchronization, a user can keep the U_MSR_PMON_CTL.ev_sel at 0x0 while enabled and write it with a valid value just prior to unfreezing the registers in other boxes.

2.1.3 Reading the Sample Interval

Software can **poll** the counters whenever it chooses, or wait to be **notified** that a counter has overflowed (by receiving a PMI).

a) **Polling** - before reading, it is recommended that software freeze the counters at either the Global level (U_MSR_PMON_GLOBAL_CTL.frz_all) or in each box with active counters (by setting *_PMON_BOX_CTL.frz to 1). After reading the event counts from the counter registers, the monitoring agent can choose to reset the event counts to avoid event-count wrap-around; or resume the counter register without resetting their values. The latter choice will require the monitoring agent to check and adjust for potential wrap-around situations.

b) **Frozen** counters - If software set the counters to freeze on overflow and send notification when it happens, the next question is: Who caused the freeze?

Overflow bits are stored hierarchically within the Intel® Xeon® Processor D-1500 Product Family uncore. First, software should read the U_MSR_PMON_GLOBAL_STATUS.ov_* bits to determine which box(es) sent an overflow. Then read that box's *_PMON_GLOBAL_STATUS.ov field to find the overflowing counter.

Note: More than one counter may overflow at any given time.

Note: Certain boxes may have more than one PMON block (e.g. IMC has a PMON block in each Channel). It may be necessary to read all STATUS registers in the box to determine which counter overflowed.

2.1.4 Enabling a New Sample Interval from Frozen Counters

a) **Clear all uncore counters**: For each box in which counting occurred, set *_PMON_BOX_CTL.rst_ctrs to 1.

b) **Clear all overflow bits**. This includes clearing U_MSR_PMON_GLOBAL_STATUS.ov_* as well as any *_BOX_STATUS registers that have their overflow bits set.

e.g. If counter 3 in Intel QPI Port 1 overflowed, software should set Q_P1_PCI_PMON_BOX_STATUS.ov[3] to 1 to clear the overflow.

c) **Create the next sample**: Reinitialize the sample by setting the monitoring data register to (2^48 - sample_interval). Or set up a new sample interval as outlined in Section 2.1.2, "Setting up a Monitoring Session".

d) **Re-enable counting**: Set U_MSR_PMON_GLOBAL_CTL.unfrz_all to 1.

2.1.5 Global Performance Monitors

Table 2-1. Global Performance Monitoring Control MSRs

MSR Name	MSR Address	Size (bits)	Description
U_MSR_PMON_GLOBAL_CONFIG	0x0702	32	UBox PMON Global Configuration
U_MSR_PMON_GLOBAL_STATUS	0x0701	32	UBox PMON Global Status
U_MSR_PMON_GLOBAL_CTL	0x0700	32	UBox PMON Global Control

2.1.5.1 Global PMON Global Control/Status Registers

The following registers represent state governing all PMUs in the uncore, both to exert global control and collect box-level information.

U_MSR_PMON_GLOBAL_CTL contains a bit that can freeze (*.frz_all*) all the uncore counters.

If an overflow is detected in any of the uncore's PMON registers, it will be summarized in U_MSR_PMON_GLOBAL_STATUS. This register accumulates overflows sent to it from the other uncore boxes. To reset these overflow bits, a user must set the corresponding bits in U_MSR_PMON_GLOBAL_STATUS to 1, which will act to clear them.

Table 2-2. U_MSR_PMON_GLOBAL_CTL Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
frz_all	31	WO	0	Freeze all uncore performance monitors.
wk_on_pmi	30	RW	0	If PMI event requested to send to core... 0 - Send event to cores already awakened 1 - Wake any sleeping core and send PMI to all cores.
unfrz_all	29	WO	0	Unfreeze all uncore performance monitors.
rsv	28:27	RV	0	Reserved. SW must write to 0 else behavior is undefined
rsv	26:18	RV	0	Reserved
pmi_core_sel	17:0	RW	0	PMI Core Select Ex: If counter overflow is sent to UBox... 000000000000000000 - No PMI sent 000000000000000001 - Send PMI to core 0 000000000001000000 - Send PMI to core 6 000000000001100010 - Send PMI to core 2, 5 & 6 etc. NOTE: If wk_on_pmi is set to 1, a wake will be sent to any sleeping core in the mask prior to sending the PMI.

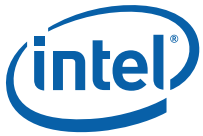


Table 2-3. U_MSR_PMON_GLOBAL_STATUS Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
rsv	63:35	RV	0	Reserved
ov_irp	34	RW1C	0	Set if overflow is detected from an IRP PMON register. NOTE: Write of '1' will clear the bit.
rsv	33:30	RV	0	Reserved
ov_rp	29	RW1C	0	Set if overflow is detected from an R2PCIe PMON register. NOTE: Write of '1' will clear the bit.
rsv	28:24	RV	0	Reserved
ov_m0	23	RW1C	0	Set if overflow is detected from an iMC0 PMON register. NOTE: Write of '1' will clear the bit.
rsv	22	RV	0	Reserved
ov_h0	21	RW1C	0	Set if overflow is detected from an HA0 PMON register. NOTE: Write of '1' will clear the bit.
rsv	20:3	RV	0	Reserved
ov_p	2	RW1C	0	Set if overflow is detected from a PCU PMON register. NOTE: Write of '1' will clear the bit.
ov_u	1	RW1C	0	Set if overflow is detected from a UBox PMON register. NOTE: Write of '1' will clear the bit.
ov_u_fixed	0	RW1C	0	Set if overflow is detected from UBox fixed PMON register. NOTE: Write of '1' will clear the bit.

Table 2-4. U_MSR_PMON_GLOBAL_CONFIG Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
rsv	31:5	RV	0	Reserved
num_c	4:0	RW	18	Number of sets of CBo PMON counters.

2.2 UBox Performance Monitoring

The UBox serves as the system configuration controller for the Intel® Xeon® Processor D-1500 Product Family.

In this capacity, the UBox acts as the central unit for a variety of functions:

- The master for reading and writing physically distributed registers across the processor using the Message Channel.
- The UBox is the intermediary for interrupt traffic, receiving interrupts from the system and dispatching interrupts to the appropriate core.
- The UBox serves as the system lock master used when quiescing the platform (e.g. bus lock).

2.2.1 UBox Performance Monitoring Overview

The UBox supports event monitoring through two programmable 48-bit wide counters (U_MSR_PMON_CTR{1:0}), and a 48-bit fixed counter which increments each U-clock. Each of these counters can be programmed (U_MSR_PMON_CTL{1:0}) to monitor any UBox event.

For information on how to setup a monitoring session, refer to Section 2.1, "Uncore Per-Socket Performance Monitoring Control".

2.2.1.1 UBox PMON Registers - On Overflow and the Consequences (PMI/Freeze)

If an overflow is detected from a UBox performance counter, the overflow bit is set at the box level (U_MSR_PMON_BOX_STATUS.ov). If its overflow enable bit (U_MSR_PMON_CTLx.ov_en) has been set to 1, the U_MSR_PMON_GLOBAL_STATUS.ov_u bit is set (see Table 2-3, "U_MSR_PMON_GLOBAL_STATUS Register – Field Definitions"), the freeze signal is broadcast to other boxes and a PMI can be generated.

Once a freeze has occurred, in order to see a new freeze, the overflow responsible for the freeze, must be cleared by setting the corresponding bit in U_MSR_PMON_BOX_STATUS.ov and U_MSR_PMON_GLOBAL_STATUs.ov_u to 1 (which acts to clear it). Assuming all the counters have been locally enabled (*.en* bit in control registers meant to monitor events) and the overflow bit(s) has been cleared, the UBox is prepared for a new sample interval. Once the global controls have been re-enabled (Section 2.1.4, "Enabling a New Sample Interval from Frozen Counters"), counting will resume.

2.2.2 UBox Performance Monitors

Table 2-5. UBox Performance Monitoring Registers (MSR)

MSR Name	MSR Address	Size (bits)	Description
U_MSR_PMON_CTR1	0x070A	64	U-Box PMON Counter 1
U_MSR_PMON_CTR0	0x0709	64	U-Box PMON Counter 0
U_MSR_PMON_BOX_STATUS	0x0708	32	U-Box PMON Box-Wide Status
U_MSR_PMON_CTL1	0x0706	64	U-Box PMON Control for Counter 1
U_MSR_PMON_CTL0	0x0705	32	U-Box PMON Control for Counter 0
U_MSR_PMON_UCLK_FIXED_CTR	0x0704	64	U-Box PMON UCLK Fixed Counter
U_MSR_PMON_UCLK_FIXED_CTL	0x0703	32	U-Box PMON UCLK Fixed Counter Control

2.2.2.1 UBox Box Level PMON State

The following registers represent the state governing all box-level PMUs in the UBox.

If an overflow is detected from one of the UBox PMON registers, the corresponding bit in the U_MSR_PMON_BOX_STATUS.ov field will be set. To reset these overflow bits, a user must write a value of '1' to them (which will clear the bits).

Table 2-6. U_MSR_PMON_BOX_STATUS Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
rsv	31:2	RV	0	Reserved
ov	1:0	RW1C	0	If an overflow is detected from the corresponding UBOX PMON register, it's overflow bit will be set. NOTE: Write of '1' will clear the bit.

2.2.2.2 UBox PMON state - Counter/Control Pairs

The following table defines the layout of the UBox performance monitor control registers. The main task of these configuration registers is to select the event to be monitored by their respective data counter (*.ev_sel*, *.umask*). Additional control bits are provided to shape the incoming events (e.g. *.invert*, *.edge_det*, *.thresh*) as well as provide additional functionality for monitoring software (*.rst*).

Table 2-7. U_MSR_PMON_CTL{1-0} Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
rsv	31:29	RV	0	Reserved
thresh	28:24	RW	0	Threshold used in counter comparison.
invert	23	RW	0	Invert comparison against Threshold. 0 - comparison will be 'is event increment >= threshold?'. 1 - comparison is inverted - 'is event increment < threshold?' NOTE: .invert is in series following .thresh, Due to this, the .thresh field must be set to a non-0 value. For events that increment by no more than 1 per cycle, set .thresh to 0x1. Also, if .edge_det is set to 1, the counter will increment when a 1 to 0 transition (i.e. falling edge) is detected.
en	22	RW	0	Local Counter Enable
rsv	21	RV	0	Reserved. SW must write to 0 else behavior is undefined.
ov_en	20	RW	0	When this bit is set to 1 and the corresponding counter overflows, a UBox overflow message is sent to the UBox's global logic. Once received, the global status register will record the overflow in U_MSR_PMON_GLOBAL_STATUS.ov_u.
rsv	19	RV	0	Reserved
edge_det	18	RW	0	When set to 1, rather than measuring the event in each cycle it is active, the corresponding counter will increment when a 0 to 1 transition (i.e. rising edge) is detected. When 0, the counter will increment in each cycle that the event is asserted. NOTE: .edge_det is in series following .thresh, Due to this, the .thresh field must be set to a non-0 value. For events that increment by no more than 1 per cycle, set .thresh to 0x1.
rst	17	WO	0	When set to 1, the corresponding counter will be cleared to 0.
rsv	16	RV	0	Reserved. SW must write to 0 else behavior is undefined.
umask	15:8	RW	0	Select subevents to be counted within the selected event.
ev_sel	7:0	RW	0	Select event to be counted.

The UBox performance monitor data registers are 48-bit wide. A counter overflow occurs when a carry out from bit 47 is detected. Software can force all uncore counting to freeze after N events by preloading a monitor with a count value of 2^{48} - N and setting the control register to send an overflow message to the global logic (*.ov_en*). During the interval of time between overflow and global disable, the counter value will wrap and continue to collect events.

If accessible, software can continuously read the data registers without disabling event collection.

Table 2-8. U_MSR_PMON_CTR{1-0} Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
rsv	63:48	RV	0	Reserved
event_count	47:0	RW-V	0	48-bit performance event counter

The UBox PMON includes a fixed counter that increments at UCLK for each cycle it is enabled.

Table 2-9. U_MSR_PMON_FIXED_CTL Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
rsv	31:23	RV	0	Reserved
en	22	RW-V	0	Enable counter when global enable is set.
rsv	21	RV	0	Reserved. SW must write to 0 else behavior is undefined.
ov_en	20	RW-V	0	When this bit is set to 1 and the corresponding counter overflows, a UBox overflow message is sent to the UBox's global logic. Once received, the global status register will record the overflow in U_MSR_PMON_GLOBAL_STATUS.ov_u_fixed.
rsv	19:0	RV	0	Reserved

Table 2-10. U_MSR_PMON_FIXED_CTR Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
rsv	63:48	RV	0	Reserved
event_count	47:0	RW-V	0	48-bit performance event counter

2.2.3 UBox Performance Monitoring Events

The set of events that can be monitored in the UBox are summarized in Section 2.2.

2.2.4 UBOX Box Events Ordered By Code

The following table summarizes the directly measured UBOX Box events.

Symbol Name	Event Code	Ctrs	Extra Select Bit	Max Inc/Cyc	Description
EVENT_MSG	0x42	0-1	0	1	VLW Received
PHOLD_CYCLES	0x45	0-1	0	1	Cycles PHOLD Assert to Ack

Symbol Name	Event Code	Ctrs	Extra Select Bit	Max Inc/Cyc	Description
RACU_REQUESTS	0x46	0-1	0	1	RACU Request

2.2.5 UBOX Box Performance Monitor Event List

The section enumerates Intel® Xeon® Processor D-1500 Product Family performance monitoring events for the UBOX Box.

EVENT_MSG

- **Title:** VLW Received
- **Category:** EVENT_MSG Events
- **Event Code:** 0x42
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** Virtual Logical Wire (legacy) message were received from Uncore. Specify the thread to filter on using NCUPMONCTRLGLCTR.ThreadID.

Table 2-11. Unit Masks for EVENT_MSG

Extension	umask [15:8]	Description
DOORBELL_RCVD	bxxxx1xxx	

PHOLD_CYCLES

- **Title:** Cycles PHOLD Assert to Ack
- **Category:** PHOLD Events
- **Event Code:** 0x45
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** PHOLD cycles. Filter from source CoreID.

Table 2-12. Unit Masks for PHOLD_CYCLES

Extension	umask [15:8]	Description
ASSERT_TO_ACK	bxxxxxxx1	Assert to ACK

RACU_REQUESTS

- **Title:** RACU Request
- **Category:** RACU Events
- **Event Code:** 0x46
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:**
- **NOTE:** This will be dropped because PHOLD is not implemented this way

2.3 Caching Agent (Cbo) Performance Monitoring

The LLC coherence engine (CBo) manages the interface between the core and the last level cache (LLC). All core transactions that access the LLC are directed from the core to a CBo via the ring interconnect. The CBo is responsible for managing data delivery from the LLC to the requesting core. It is also responsible for maintaining coherence between the cores within the socket that share the LLC; generating snoops and collecting snoop responses from the local cores when the MESIF protocol requires it.

So, if the CBo fielding the core request indicates that a core within the socket owns the line (for a coherent read), the request is snooped to that local core. That same CBo will then snoop all peers which might have the address cached (other cores, remote sockets, etc.) and send the request to the appropriate Home Agent for conflict checking, memory requests and writebacks.

The CBo manages local conflicts by ensuring that only one request is issued to the system for a specific cacheline.

The Intel® Xeon® Processor D-1500 Product Family uncore contains up to 8 instances of the CBo, each assigned to manage a (up to) distinct 1.5MB slice of the processor's total LLC capacity. A slice that can be up to 12-way set associative. For processors with fewer than 8 1.5MB LLC slices, the CBo Boxes or missing slices will still be active and track ring traffic caused by their co-located core even if they have no LLC related traffic to track (i.e. hits/misses/snoops).

Every physical memory address in the system is uniquely associated with a single CBo instance via a proprietary hashing algorithm that is designed to keep the distribution of traffic across the CBo instances relatively uniform for a wide range of possible address patterns. This enables the individual CBo instances to operate independently, each managing its slice of the physical address space without any CBo in a given socket ever needing to communicate with the other CBos in that same socket.

2.3.1 CBo Performance Monitoring Overview

Each of the CBos in the Intel® Xeon® Processor D-1500 Product Family uncore supports event monitoring through four 48-bit wide counters (Cn_MSR_PMON_CTR{3:0}). With but a small number of exceptions, each of these counters can be programmed (Cn_MSR_PMON_CTL{3:0}) for any available event.

NOTE: Occupancy Events can only be measured in Counter 0.

CBo counter 0 can increment by a maximum of 20 per cycle; counters 1-3 can increment by 1 per cycle.

Some uncore performance events that monitor transaction activities require additional details that must be programmed in a filter register. Each Cbo provides two filter registers and allows only one such event to be programmed at a given time, see Section 2.3.2.3.

For information on how to setup a monitoring session, refer to Section 2.1, "Uncore Per-Socket Performance Monitoring Control".

2.3.1.1 Special Note on CBo Occupancy Events

Although only counter 0 supports occupancy events, it is possible to program counters 1-3 to monitor the same occupancy event by selecting the "OCCUPANCY_COUNTER0" event code on counters 1-3.

This allows:

Thresholding

2.3.1.2 CBo PMON Registers - On Overflow and the Consequences (PMI/Freeze)

If an overflow is detected from a CBo performance counter, the overflow bit is set at the box level (Cn_MSR_PMON_BOX_STATUS.ov)

2.3.2 CBo Performance Monitors

Note: The number of CBoxes varies with the number of Cores in a system. To determine the number of CBoxes, SW should read bits 15:0 in the CAPID5 register located at Device 30, Function 3, Offset 0x98. These 16 bits form a bit vector of available LLC slices and the CBoxes that manage those slices. For example: If bits 15:0 read 0x0F0F, the PMON blocks corresponding to CBoxes 0-3 and 8-11 are available and CBoxes 4-7 and 12-15 are not available.

Table 2-13. CBo Performance Monitoring Registers (MSR) (Sheet 1 of 8)

MSR Name	MSR Address	Size (bits)	Description
CBo 0 PMON Registers			
Generic Counters			
C0_MSR_PMON_CTR3	0x0E0B	64	CBo 0 PMON Counter 3
C0_MSR_PMON_CTR2	0x0E0A	64	CBo 0 PMON Counter 2
C0_MSR_PMON_CTR1	0x0E09	64	CBo 0 PMON Counter 1
C0_MSR_PMON_CTR0	0x0E08	64	CBo 0 PMON Counter 0
Box-Level Filter			
C0_MSR_PMON_BOX_FILTER1	0x0E06	32	CBo 0 PMON Filter1
C0_MSR_PMON_BOX_FILTER0	0x0E05	32	CBo 0 PMON Filter0
Generic Counter Control			
C0_MSR_PMON_CTL3	0x0E04	32	CBo 0 PMON Control for Counter 3
C0_MSR_PMON_CTL2	0x0E03	32	CBo 0 PMON Control for Counter 2
C0_MSR_PMON_CTL1	0x0E02	32	CBo 0 PMON Control for Counter 1
C0_MSR_PMON_CTL0	0x0E01	32	CBo 0 PMON Control for Counter 0
Box-Level Control/Status			
C0_MSR_PMON_BOX_STATUS	0x0E07	32	CBo 0 PMON Box-Wide Status
C0_MSR_PMON_BOX_CTL	0x0E00	32	CBo 0 PMON Box-Wide Control
CBo 1 PMON Registers			
Generic Counters			
C1_MSR_PMON_CTR3	0x0E1B	64	CBo 1 PMON Counter 3
C1_MSR_PMON_CTR2	0x0E1A	64	CBo 1 PMON Counter 2
C1_MSR_PMON_CTR1	0x0E19	64	CBo 1 PMON Counter 1
C1_MSR_PMON_CTR0	0x0E18	64	CBo 1 PMON Counter 0
Box-Level Filter			
C1_MSR_PMON_BOX_FILTER1	0x0E16	32	CBo 1 PMON Filter1
C1_MSR_PMON_BOX_FILTER0	0x0E15	32	CBo 1 PMON Filter0
Generic Counter Control			
C1_MSR_PMON_CTL3	0x0E14	32	CBo 1 PMON Control for Counter 3

Table 2-13. CBo Performance Monitoring Registers (MSR) (Sheet 2 of 8)

MSR Name	MSR Address	Size (bits)	Description
C1_MSR_PMON_CTL2	0x0E13	32	CBo 1 PMON Control for Counter 2
C1_MSR_PMON_CTL1	0x0E12	32	CBo 1 PMON Control for Counter 1
C1_MSR_PMON_CTL0	0x0E11	32	CBo 1 PMON Control for Counter 0
Box-Level Control/Status			
C1_MSR_PMON_BOX_STATUS	0x0E17	32	CBo 1 PMON Box-Wide Status
C1_MSR_PMON_BOX_CTL	0x0E10	32	CBo 1 PMON Box-Wide Control
CBo 2 PMON Registers			
Generic Counters			
C2_MSR_PMON_CTR3	0x0E2B	64	CBo 2 PMON Counter 3
C2_MSR_PMON_CTR2	0x0E2A	64	CBo 2 PMON Counter 2
C2_MSR_PMON_CTR1	0x0E29	64	CBo 2 PMON Counter 1
C2_MSR_PMON_CTR0	0x0E28	64	CBo 2 PMON Counter 0
Box-Level Filter			
C2_MSR_PMON_BOX_FILTER1	0x0E26	32	CBo 2 PMON Filter1
C2_MSR_PMON_BOX_FILTER0	0x0E25	32	CBo 2 PMON Filter0
Generic Counter Control			
C2_MSR_PMON_CTL3	0x0E24	32	CBo 2 PMON Control for Counter 3
C2_MSR_PMON_CTL2	0x0E23	32	CBo 2 PMON Control for Counter 2
C2_MSR_PMON_CTL1	0x0E22	32	CBo 2 PMON Control for Counter 1
C2_MSR_PMON_CTL0	0x0E21	32	CBo 2 PMON Control for Counter 0
Box-Level Control/Status			
C2_MSR_PMON_BOX_STATUS	0x0E27	32	CBo 2 PMON Box-Wide Status
C2_MSR_PMON_BOX_CTL	0x0E20	32	CBo 2 PMON Box-Wide Control
CBo 3 PMON Registers			
Generic Counters			
C3_MSR_PMON_CTR3	0x0E3B	64	CBo 3 PMON Counter 3
C3_MSR_PMON_CTR2	0x0E3A	64	CBo 3 PMON Counter 2
C3_MSR_PMON_CTR1	0x0E39	64	CBo 3 PMON Counter 1
C3_MSR_PMON_CTR0	0x0E38	64	CBo 3 PMON Counter 0
Box-Level Filter			
C3_MSR_PMON_BOX_FILTER1	0x0E36	32	CBo 3 PMON Filter1
C3_MSR_PMON_BOX_FILTER0	0x0E35	32	CBo 3 PMON Filter0
Generic Counter Control			
C3_MSR_PMON_CTL3	0x0E34	32	CBo 3 PMON Control for Counter 3
C3_MSR_PMON_CTL2	0x0E33	32	CBo 3 PMON Control for Counter 2
C3_MSR_PMON_CTL1	0x0E32	32	CBo 3 PMON Control for Counter 1
C3_MSR_PMON_CTL0	0x0E31	32	CBo 3 PMON Control for Counter 0
Box-Level Control/Status			
C3_MSR_PMON_BOX_STATUS	0x0E37	32	CBo 3 PMON Box-Wide Status

Table 2-13. CBo Performance Monitoring Registers (MSR) (Sheet 3 of 8)

MSR Name	MSR Address	Size (bits)	Description
C3_MSR_PMON_BOX_CTL	0x0E30	32	CBo 3 PMON Box-Wide Control
CBo 4 PMON Registers			
Generic Counters			
C4_MSR_PMON_CTR3	0x0E4B	64	CBo 4 PMON Counter 3
C4_MSR_PMON_CTR2	0x0E4A	64	CBo 4 PMON Counter 2
C4_MSR_PMON_CTR1	0x0E49	64	CBo 4 PMON Counter 1
C4_MSR_PMON_CTR0	0x0E48	64	CBo 4 PMON Counter 0
Box-Level Filter			
C4_MSR_PMON_BOX_FILTER1	0x0E46	32	CBo 4 PMON Filter1
C4_MSR_PMON_BOX_FILTER0	0x0E45	32	CBo 4 PMON Filter0
Generic Counter Control			
C4_MSR_PMON_CTL3	0x0E44	32	CBo 4 PMON Control for Counter 3
C4_MSR_PMON_CTL2	0x0E43	32	CBo 4 PMON Control for Counter 2
C4_MSR_PMON_CTL1	0x0E42	32	CBo 4 PMON Control for Counter 1
C4_MSR_PMON_CTL0	0x0E41	32	CBo 4 PMON Control for Counter 0
Box-Level Control/Status			
C4_MSR_PMON_BOX_STATUS	0x0E47	32	CBo 4 PMON Box-Wide Status
C4_MSR_PMON_BOX_CTL	0x0E40	32	CBo 4 PMON Box-Wide Control
CBo 5 PMON Registers			
Generic Counters			
C5_MSR_PMON_CTR3	0x0E5B	64	CBo 5 PMON Counter 3
C5_MSR_PMON_CTR2	0x0E5A	64	CBo 5 PMON Counter 2
C5_MSR_PMON_CTR1	0x0E59	64	CBo 5 PMON Counter 1
C5_MSR_PMON_CTR0	0x0E58	64	CBo 5 PMON Counter 0
Box-Level Filter			
C5_MSR_PMON_BOX_FILTER1	0x0E56	32	CBo 5 PMON Filter1
C5_MSR_PMON_BOX_FILTER0	0x0E55	32	CBo 5 PMON Filter0
Generic Counter Control			
C5_MSR_PMON_CTL3	0x0E54	32	CBo 5 PMON Control for Counter 3
C5_MSR_PMON_CTL2	0x0E53	32	CBo 5 PMON Control for Counter 2
C5_MSR_PMON_CTL1	0x0E52	32	CBo 5 PMON Control for Counter 1
C5_MSR_PMON_CTL0	0x0E51	32	CBo 5 PMON Control for Counter 0
Box-Level Control/Status			
C5_MSR_PMON_BOX_STATUS	0x0E57	32	CBo 5 PMON Box-Wide Status
C5_MSR_PMON_BOX_CTL	0x0E50	32	CBo 5 PMON Box-Wide Control
CBo 6 PMON Registers			
Generic Counters			
C6_MSR_PMON_CTR3	0x0E6B	64	CBo 6 PMON Counter 3

Table 2-13. CBo Performance Monitoring Registers (MSR) (Sheet 4 of 8)

MSR Name	MSR Address	Size (bits)	Description
C6_MSR_PMON_CTR2	0x0E6A	64	CBo 6 PMON Counter 2
C6_MSR_PMON_CTR1	0x0E69	64	CBo 6 PMON Counter 1
C6_MSR_PMON_CTR0	0x0E68	64	CBo 6 PMON Counter 0
Box-Level Filter			
C6_MSR_PMON_BOX_FILTER1	0x0E66	32	CBo 6 PMON Filter1
C6_MSR_PMON_BOX_FILTER0	0x0E65	32	CBo 6 PMON Filter0
Generic Counter Control			
C6_MSR_PMON_CTL3	0x0E64	32	CBo 6 PMON Control for Counter 3
C6_MSR_PMON_CTL2	0x0E63	32	CBo 6 PMON Control for Counter 2
C6_MSR_PMON_CTL1	0x0E62	32	CBo 6 PMON Control for Counter 1
C6_MSR_PMON_CTL0	0x0E61	32	CBo 6 PMON Control for Counter 0
Box-Level Control/Status			
C6_MSR_PMON_BOX_STATUS	0x0E67	32	CBo 6 PMON Box-Wide Status
C6_MSR_PMON_BOX_CTL	0x0E60	32	CBo 6 PMON Box-Wide Control
CBo 7 PMON Registers			
Generic Counters			
C7_MSR_PMON_CTR3	0x0E7B	64	CBo 7 PMON Counter 3
C7_MSR_PMON_CTR2	0x0E7A	64	CBo 7 PMON Counter 2
C7_MSR_PMON_CTR1	0x0E79	64	CBo 7 PMON Counter 1
C7_MSR_PMON_CTR0	0x0E78	64	CBo 7 PMON Counter 0
Box-Level Filter			
C7_MSR_PMON_BOX_FILTER1	0x0E76	32	CBo 7 PMON Filter1
C7_MSR_PMON_BOX_FILTER0	0x0E75	32	CBo 7 PMON Filter0
Generic Counter Control			
C7_MSR_PMON_CTL3	0x0E74	32	CBo 7 PMON Control for Counter 3
C7_MSR_PMON_CTL2	0x0E73	32	CBo 7 PMON Control for Counter 2
C7_MSR_PMON_CTL1	0x0E72	32	CBo 7 PMON Control for Counter 1
C7_MSR_PMON_CTL0	0x0E71	32	CBo 7 PMON Control for Counter 0
Box-Level Control/Status			
C7_MSR_PMON_BOX_STATUS	0x0E77	32	CBo 7 PMON Box-Wide Status
C7_MSR_PMON_BOX_CTL	0x0E70	32	CBo 7 PMON Box-Wide Control
CBo 8 PMON Registers			
Generic Counters			
C8_MSR_PMON_CTR3	0x0E8B	64	CBo 8 PMON Counter 3
C8_MSR_PMON_CTR2	0x0E8A	64	CBo 8 PMON Counter 2
C8_MSR_PMON_CTR1	0x0E89	64	CBo 8 PMON Counter 1
C8_MSR_PMON_CTR0	0x0E88	64	CBo 8 PMON Counter 0
Box-Level Filter			
C8_MSR_PMON_BOX_FILTER1	0x0E86	32	CBo 8 PMON Filter1

Table 2-13. CBo Performance Monitoring Registers (MSR) (Sheet 5 of 8)

MSR Name	MSR Address	Size (bits)	Description
C8_MSR_PMON_BOX_FILTER0	0x0E85	32	CBo 8 PMON Filter0
Generic Counter Control			
C8_MSR_PMON_CTL3	0x0E84	32	CBo 8 PMON Control for Counter 3
C8_MSR_PMON_CTL2	0x0E83	32	CBo 8 PMON Control for Counter 2
C8_MSR_PMON_CTL1	0x0E82	32	CBo 8 PMON Control for Counter 1
C8_MSR_PMON_CTL0	0x0E81	32	CBo 8 PMON Control for Counter 0
Box-Level Control/Status			
C8_MSR_PMON_BOX_STATUS	0x0E87	32	CBo 8 PMON Box-Wide Status
C8_MSR_PMON_BOX_CTL	0x0E80	32	CBo 8 PMON Box-Wide Control
CBo 9 PMON Registers			
Generic Counters			
C9_MSR_PMON_CTR3	0x0E9B	64	CBo 9 PMON Counter 3
C9_MSR_PMON_CTR2	0x0E9A	64	CBo 9 PMON Counter 2
C9_MSR_PMON_CTR1	0x0E99	64	CBo 9 PMON Counter 1
C9_MSR_PMON_CTR0	0x0E98	64	CBo 9 PMON Counter 0
Box-Level Filter			
C9_MSR_PMON_BOX_FILTER1	0x0E96	32	CBo 9 PMON Filter1
C9_MSR_PMON_BOX_FILTER0	0x0E95	32	CBo 9 PMON Filter0
Generic Counter Control			
C9_MSR_PMON_CTL3	0x0E94	32	CBo 9 PMON Control for Counter 3
C9_MSR_PMON_CTL2	0x0E93	32	CBo 9 PMON Control for Counter 2
C9_MSR_PMON_CTL1	0x0E92	32	CBo 9 PMON Control for Counter 1
C9_MSR_PMON_CTL0	0x0E91	32	CBo 9 PMON Control for Counter 0
Box-Level Control/Status			
C9_MSR_PMON_BOX_STATUS	0x0E97	32	CBo 9 PMON Box-Wide Status
C9_MSR_PMON_BOX_CTL	0x0E90	32	CBo 9 PMON Box-Wide Control
CBo 10 PMON Registers			
Generic Counters			
C10_MSR_PMON_CTR3	0x0EAB	64	CBo 10 PMON Counter 3
C10_MSR_PMON_CTR2	0x0EAA	64	CBo 10 PMON Counter 2
C10_MSR_PMON_CTR1	0x0EA9	64	CBo 10 PMON Counter 1
C10_MSR_PMON_CTR0	0x0EA8	64	CBo 10 PMON Counter 0
Box-Level Filter			
C10_MSR_PMON_BOX_FILTER1	0x0EA6	32	CBo 10 PMON Filter1
C10_MSR_PMON_BOX_FILTER0	0x0EA5	32	CBo 10 PMON Filter0
Generic Counter Control			
C10_MSR_PMON_CTL3	0x0EA4	32	CBo 10 PMON Control for Counter 3
C10_MSR_PMON_CTL2	0x0EA3	32	CBo 10 PMON Control for Counter 2
C10_MSR_PMON_CTL1	0x0EA2	32	CBo 10 PMON Control for Counter 1

Table 2-13. CBo Performance Monitoring Registers (MSR) (Sheet 6 of 8)

MSR Name	MSR Address	Size (bits)	Description
C10_MSR_PMON_CTL0	0x0EA1	32	CBo 10 PMON Control for Counter 0
Box-Level Control/Status			
C10_MSR_PMON_BOX_STATUS	0x0EA7	32	CBo 10 PMON Box-Wide Status
C10_MSR_PMON_BOX_CTL	0x0EA0	32	CBo 10 PMON Box-Wide Control
CBo 11 PMON Registers			
Generic Counters			
C11_MSR_PMON_CTR3	0x0EBB	64	CBo 11 PMON Counter 3
C11_MSR_PMON_CTR2	0x0EBA	64	CBo 11 PMON Counter 2
C11_MSR_PMON_CTR1	0x0EB9	64	CBo 11 PMON Counter 1
C11_MSR_PMON_CTR0	0x0EB8	64	CBo 11 PMON Counter 0
Box-Level Filter			
C11_MSR_PMON_BOX_FILTER1	0x0EB6	32	CBo 11 PMON Filter1
C11_MSR_PMON_BOX_FILTER0	0x0EB5	32	CBo 11 PMON Filter0
Generic Counter Control			
C11_MSR_PMON_CTL3	0x0EB4	32	CBo 11 PMON Control for Counter 3
C11_MSR_PMON_CTL2	0x0EB3	32	CBo 11 PMON Control for Counter 2
C11_MSR_PMON_CTL1	0x0EB2	32	CBo 11 PMON Control for Counter 1
C11_MSR_PMON_CTL0	0x0EB1	32	CBo 11 PMON Control for Counter 0
Box-Level Control/Status			
C11_MSR_PMON_BOX_STATUS	0x0EB7	32	CBo 11 PMON Box-Wide Status
C11_MSR_PMON_BOX_CTL	0x0EB0	32	CBo 11 PMON Box-Wide Control
CBo 12 PMON Registers			
Generic Counters			
C12_MSR_PMON_CTR3	0x0ECB	64	CBo 12 PMON Counter 3
C12_MSR_PMON_CTR2	0x0ECA	64	CBo 12 PMON Counter 2
C12_MSR_PMON_CTR1	0x0EC9	64	CBo 12 PMON Counter 1
C12_MSR_PMON_CTR0	0x0EC8	64	CBo 12 PMON Counter 0
Box-Level Filter			
C12_MSR_PMON_BOX_FILTER1	0x0EC6	32	CBo 12 PMON Filter1
C12_MSR_PMON_BOX_FILTER0	0x0EC5	32	CBo 12 PMON Filter0
Generic Counter Control			
C12_MSR_PMON_CTL3	0x0EC4	32	CBo 12 PMON Control for Counter 3
C12_MSR_PMON_CTL2	0x0EC3	32	CBo 12 PMON Control for Counter 2
C12_MSR_PMON_CTL1	0x0EC2	32	CBo 12 PMON Control for Counter 1
C12_MSR_PMON_CTL0	0x0EC1	32	CBo 12 PMON Control for Counter 0
Box-Level Control/Status			
C12_MSR_PMON_BOX_STATUS	0x0EC7	32	CBo 12 PMON Box-Wide Status
C12_MSR_PMON_BOX_CTL	0x0EC0	32	CBo 12 PMON Box-Wide Control

Table 2-13. CBo Performance Monitoring Registers (MSR) (Sheet 7 of 8)

MSR Name	MSR Address	Size (bits)	Description
CBo 13 PMON Registers			
Generic Counters			
C13_MSR_PMON_CTR3	0x0EDB	64	CBo 13 PMON Counter 3
C13_MSR_PMON_CTR2	0x0EDA	64	CBo 13 PMON Counter 2
C13_MSR_PMON_CTR1	0x0ED9	64	CBo 13 PMON Counter 1
C13_MSR_PMON_CTR0	0x0ED8	64	CBo 13 PMON Counter 0
Box-Level Filter			
C13_MSR_PMON_BOX_FILTER1	0x0ED6	32	CBo 13 PMON Filter1
C13_MSR_PMON_BOX_FILTER0	0x0ED5	32	CBo 13 PMON Filter0
Generic Counter Control			
C13_MSR_PMON_CTL3	0x0ED4	32	CBo 13 PMON Control for Counter 3
C13_MSR_PMON_CTL2	0x0ED3	32	CBo 13 PMON Control for Counter 2
C13_MSR_PMON_CTL1	0x0ED2	32	CBo 13 PMON Control for Counter 1
C13_MSR_PMON_CTL0	0x0ED1	32	CBo 13 PMON Control for Counter 0
Box-Level Control/Status			
C13_MSR_PMON_BOX_STATUS	0x0ED7	32	CBo 13 PMON Box-Wide Status
C13_MSR_PMON_BOX_CTL	0x0ED0	32	CBo 13 PMON Box-Wide Control
CBo 14 PMON Registers			
Generic Counters			
C14_MSR_PMON_CTR3	0x0EEB	64	CBo 14 PMON Counter 3
C14_MSR_PMON_CTR2	0x0EEA	64	CBo 14 PMON Counter 2
C14_MSR_PMON_CTR1	0x0EE9	64	CBo 14 PMON Counter 1
C14_MSR_PMON_CTR0	0x0EE8	64	CBo 14 PMON Counter 0
Box-Level Filter			
C14_MSR_PMON_BOX_FILTER1	0x0EE6	32	CBo 14 PMON Filter1
C14_MSR_PMON_BOX_FILTER0	0x0EE5	32	CBo 14 PMON Filter0
Generic Counter Control			
C14_MSR_PMON_CTL3	0x0EE4	32	CBo 14 PMON Control for Counter 3
C14_MSR_PMON_CTL2	0x0EE3	32	CBo 14 PMON Control for Counter 2
C14_MSR_PMON_CTL1	0x0EE2	32	CBo 14 PMON Control for Counter 1
C14_MSR_PMON_CTL0	0x0EE1	32	CBo 14 PMON Control for Counter 0
Box-Level Control/Status			
C14_MSR_PMON_BOX_STATUS	0x0EE7	32	CBo 14 PMON Box-Wide Status
C14_MSR_PMON_BOX_CTL	0x0EE0	32	CBo 14 PMON Box-Wide Control
CBo 15 PMON Registers			
Generic Counters			
C15_MSR_PMON_CTR3	0x0EFB	64	CBo 15 PMON Counter 3
C15_MSR_PMON_CTR2	0x0EFA	64	CBo 15 PMON Counter 2
C15_MSR_PMON_CTR1	0x0EF9	64	CBo 15 PMON Counter 1

Table 2-13. CBo Performance Monitoring Registers (MSR) (Sheet 8 of 8)

MSR Name	MSR Address	Size (bits)	Description
C15_MSR_PMON_CTR0	0x0EF8	64	CBo 15 PMON Counter 0
Box-Level Filter			
C15_MSR_PMON_BOX_FILTER1	0x0EF6	32	CBo 15 PMON Filter1
C15_MSR_PMON_BOX_FILTER0	0x0EF5	32	CBo 15 PMON Filter0
Generic Counter Control			
C15_MSR_PMON_CTL3	0x0EF4	32	CBo 15 PMON Control for Counter 3
C15_MSR_PMON_CTL2	0x0EF3	32	CBo 15 PMON Control for Counter 2
C15_MSR_PMON_CTL1	0x0EF2	32	CBo 15 PMON Control for Counter 1
C15_MSR_PMON_CTL0	0x0EF1	32	CBo 15 PMON Control for Counter 0
Box-Level Control/Status			
C15_MSR_PMON_BOX_STATUS	0x0EF7	32	CBo 15 PMON Box-Wide Status
C15_MSR_PMON_BOX_CTL	0x0EF0	32	CBo 15 PMON Box-Wide Control

2.3.2.1 CBo Box Level PMON State

The following registers represent the state governing all box-level PMUs in the CBo.

In the case of the CBo, the Cn_MSR_PMON_BOX_CTL register provides the ability to manually freeze the counters in the box (*.frz*) and reset the generic state (*.rst_ctrs* and *.rst_ctrl*).

If an overflow is detected from one of the CBo PMON registers, the corresponding bit in the Cn_MSR_PMON_BOX_STATUS.ov field will be set. To reset these overflow bits, a user must write a value of '1' to them (which will clear the bits).

Table 2-14. Cn_MSR_PMON_BOX_CTL Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
rsv	31:18	RV	0	Reserved
rsv	17:16	RV	0	Reserved; SW must write to 1 else behavior is undefined.
rsv	15:9	RV	0	Reserved
frz	8	WO	0	Freeze. If set to 1 the counters in this box will be frozen.
rsv	7:2	RV	0	Reserved
rst_ctrs	1	WO	0	Reset Counters. When set to 1, the Counter Registers will be reset to 0.
rst_ctrl	0	WO	0	Reset Control. When set to 1, the Counter Control Registers will be reset to 0.

Table 2-15. Cn_MSR_PMON_BOX_STATUS Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
rsv	31:4	RV	0	Reserved
ov	3:0	RW1C	0	If an overflow is detected from the corresponding Cn_MSR_PMON_CTR register, it's overflow bit will be set. NOTE: Write of '1' will clear the bit.

2.3.2.2 CBo PMON state - Counter/Control Pairs

The following table defines the layout of the CBo performance monitor control registers. The main task of these configuration registers is to select the event to be monitored by their respective data counter (*.ev_sel*, *.umask*). Additional control bits are provided to shape the incoming events (e.g. *.invert*, *.edge_det*, *.thresh*) as well as provide additional functionality for monitoring software (*.rst*).

Table 2-16. Cn_MSR_PMON_CTL{3-0} Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
thresh	31:24	RW-V	0	Threshold used in counter comparison.
invert	23	RW-V	0	Invert comparison against Threshold. 0 - comparison will be 'is event increment >= threshold?'. 1 - comparison is inverted - 'is event increment < threshold?' NOTE: .invert is in series following .thresh, Due to this, the .thresh field must be set to a non-0 value. For events that increment by no more than 1 per cycle, set .thresh to 0x1. Also, if .edge_det is set to 1, the counter will increment when a 1 to 0 transition (i.e. falling edge) is detected.
en	22	RW-V	0	Local Counter Enable.
rsv	21:20	RV	0	Reserved; SW must write to 0 else behavior is undefined.
tid_en	19	RW-V	0	TID Filter Enable
edge_det	18	RW-V	0	When set to 1, rather than measuring the event in each cycle it is active, the corresponding counter will increment when a 0 to 1 transition (i.e. rising edge) is detected. When 0, the counter will increment in each cycle that the event is asserted. NOTE: .edge_det is in series following .thresh, Due to this, the .thresh field must be set to a non-0 value. For events that increment by no more than 1 per cycle, set .thresh to 0x1.
rst	17	WO	0	When set to 1, the corresponding counter will be cleared to 0.
rsv	16	RV	0	Reserved. SW must write to 0 else behavior is undefined.
umask	15:8	RW-V	0	Select subevents to be counted within the selected event.
ev_sel	7:0	RW-V	0	Select event to be counted.

The CBo performance monitor data registers are 48b wide. A counter overflow occurs when a carry out from bit 47 is detected. Software can force all uncore counting to freeze after N events by preloading a monitor with a count value of 2^{48} - N and setting the control register to send an overflow message to the UBox (refer to Section 2.1.1, "Counter Overflow"). During the interval of time between overflow and global disable, the counter value will wrap and continue to collect events.

If accessible, software can continuously read the data registers without disabling event collection.

Table 2-17. Cn_MSR_PMON_CTR{3-0} Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
rsv	63:48	RV	0	Reserved
event_count	47:0	RW-V	0	48-bit performance event counter

2.3.2.3 CBo Filter Registers (Cn_MSR_PMON_BOX_FILTER{0,1})

In addition to generic event counting, each CBo provides a pair of FILTER registers that allow a user to filter various traffic as it applies to specific events (see Event Section for more information). LLC_LOOKUP may be filtered by the cacheline state, while TOR_INSERTS and TOR_OCCUPANCY may be filtered by the opcode of the queued request as well as the corresponding NodeID.

Any of the CBo events may be filtered by Thread/Core-ID. To do so, the control register's .*tid_en* bit must be set to 1 and the tid field in the FILTER register filled out.

Note: Only one of these filtering criteria may be applied at a time.

Table 2-18. Cn_MSR_PMON_BOX_FILTER0 Register – Field Definitions

Field	Bits	Atrtr	HW Reset Val	Description
rsv	31:24	RV	0	Reserved SW must set to 0 else behavior is undefined
state	23:17	RW	0	Select state to monitor for LLC_LOOKUP event. Setting multiple bits in this field will allow a user to track multiple states. b1xxxxxx - M'state. bx1xxxxx - D state. bxx1xxxx - F state. bxxx1xxx - M state bxxxx1xx - E state. bxxxxx1x - S state. bxxxxxx1 - I state.
rsv	16:6	RV	0	Reserved SW must set to 0 else behavior is undefined
tid	5:0	0	0	[5] Non-thread related data [4:1] Core-ID [0] Thread 1/0 When .tid_en is 0; the specified counter will count ALL events. On BDX, 'ALL events' includes traffic from the CPU (e.g. Data Reqeusts) and IIO. Thread-ID 0x3F is reserved for non-associated requests such as: - LLC victims - PMSeq - External Snoops

Table 2-19. Cn_MSR_PMON_BOX_FILTER1 Register – Field Definitions

Field	Bits	Atrtr	HW Reset Val	Description
isoc	31	RW	0	Match on ISOC Requests
nc	30	RW	0	Match on Non-Coherent Requests
rsv	29	RV	0	Reserved; SW must write else behavior is undefined
opc (7b IDI Opcode w/top 2b 0x3)	28:20	RW	0	Match on Opcode (see Table 2-14, "Cn_MSR_PMON_BOX_CTL Register – Field Definitions") NOTE: Only tracks opcodes that come from the IRQ. It is not possible to track snoops (from IPQ) or other transactions from the ISMQ.
rsv	19:15	RV	0	Reserved
nid	15:0	RW	0	Match on Target NodeID

Table 2-20. Opcode Match by IDI Packet Type for Cn_MSR_PMON_BOX_FILTER.opc (Sheet 1 of 2)

opc Value	Opcode	Defn
0x180	RFO	Demand Data RFO - Read for Ownership requests from core for lines to be cached in E
0x181	CRd	Demand Code Read - Full cache-line read requests from core for lines to be cached in S, typically for code
0x182	DRd	Demand Data Read - Full cache-line read requests from core for lines to be cached in S or E, typically for data
0x187	PRd	Partial Reads (UC) - Partial read requests of 0-32B (IIO can be up to 64B). Uncacheable.
0x18C	WCiLF	Streaming Store - Full - Write invalidate for full cache line of write combining stores
0x18D	WCiL	Streaming Store - Partial - Write invalidate for partial cache line of write combining stores
0x190	PrefRFO	Prefetch RFO into LLC but don't pass to L2. Includes Hints
0x191	PrefCode	Prefetch Code into LLC but don't pass to L2. Includes Hints
0x192	PrefData	Prefetch Data into LLC but don't pass to L2. Includes Hints
0x193	PCIWiL	PCIe Write (full - non-allocating) - Partial line MMIO write transactions from IIO (P2P). Not used for coherent transactions. Uncacheable.
0x194	PCIWiLF	PCIe Write (partial - non-allocating) - Full line MMIO write transactions from IIO (P2P). Not used for coherent transactions. Uncacheable.
0x19C	PCIItoM	PCIe Write (allocating) - Similar to ItoM - requests exclusive ownership but does not require data read and IIO does not guarantee it will modify line
0x19E	PCIRdCur	PCIe read current - Read Current requests from IIO. Used to read data without changing state.
0x1C4	WbMtoI	Request writeback Modified invalidate line - Evict full M-state cache line from core. Guarantees core has no cached copies.

Table 2-20. Opcode Match by IDI Packet Type for Cn_MSR_PMON_BOX_FILTER.opc (Sheet 2 of 2)

opc Value	Opcode	Defn
0x1C5	WbMtoE	Request writeback Modified set to Exclusive - Evict full M-state cache line from core.
0x1C8	ItoM	Request Invalidate Line - Request Exclusive Ownership of cache line
0x1E4	PCINSRd	PCIe Non-Snoop Read - Non-snoop read requests of full cache lines from IIO. (SW must guarantee coherency)
0x1E5	PCINSWr	PCIe Non-Snoop Write (partial) - Non-snoop write requests of partial cache lines from IIO. Always uncacheable.
0x1E6	PCINSWrF	PCIe Non-Snoop Write (full) - Non-snoop write requests of full cache lines from IIO. Always uncacheable.

2.3.3 CBo Performance Monitoring Events

The performance monitoring events within the CBo include all events internal to the LLC as well as events which track ring related activity at the CBo/Core ring stops.

CBo performance monitoring events can be used to track LLC access rates, LLC hit/miss rates, LLC eviction and fill rates, and to detect evidence of back pressure on the LLC pipelines. In addition, the CBo has performance monitoring events for tracking MESI state transitions that occur as a result of data sharing across sockets in a multi-socket system. And finally, there are events in the CBo for tracking ring traffic at the CBo/Core sink inject points.

Every event in the CBo is from the point of view of the LLC and is not associated with any specific core since all cores in the socket send their LLC transactions to all CBos in the socket. However, the CBo provides a thread-id field in the Cn_MSR_PMON_BOX_FILTER register which can be applied to the CBo events to obtain the interactions between specific cores and threads.

There are separate sets of counters for each CBo instance. For any event, to get an aggregate count of that event for the entire LLC, the counts across the CBo instances must be added together. The counts can be averaged across the CBo instances to get a view of the typical count of an event from the perspective of the individual CBos. Individual per-CBo deviations from the average can be used to identify hot-spotting across the CBos or other evidences of non-uniformity in LLC behavior across the CBos. Such hot-spotting should be rare, though a repetitive polling on a fixed physical address is one obvious example of a case where an analysis of the deviations across the CBos would indicate hot-spotting.

2.3.3.1 Acronyms frequently used in CBo Events

The Rings:

AD (Address) Ring - Core Read/Write Requests and Intel QPI Snoops. Carries Intel QPI requests and snoop responses from C to Intel QPI.

BL (Block or Data) Ring - Data == 2 transfers for 1 cache line

AK (Acknowledge) Ring - Acknowledges Intel QPI to CBo and CBo to Core. Carries snoop responses from Core to CBo.

IV (Invalidate) Ring - CBo Snoop requests of core caches

Internal CBo Queues:

IRQ - Ingress Request Queue on AD Ring. Associated with requests from core.

IPQ - Ingress Probe Queue on AD Ring. Associated with snoops from Intel QPI LL.

ISMQ - Ingress Subsequent Messages (response queue). Associated with messages responses to ingress requests (e.g. data responses, Intel QPI complete messages, core snoop response messages and GO reset queue).

TOR - Table Of Requests. Tracks pending CBo transactions.

QPI_IGR - Intel QPI credits for AD or BL ring. Credits to access the Intel QPI are necessary to broadcast snoops.

RxR (aka IGR) /TxR (aka EGR) - Ingress (requests from the Cores) and Egress (requests headed for the Ring) queues

2.3.3.2 The Queues

There are several internal occupancy queue counters, each of which is 5bits wide and dedicated to its queue: IRQ, IPQ, ISMQ, QPI_IGR, IGR, EGR and the TOR.

2.3.4 CBO Box Events Ordered By Code

The following table summarizes the directly measured CBO Box events.

Symbol Name	Event Code	Ctrs	Max Inc/Cyc	Description
CLOCKTICKS	0x00	0-3	1	Uncore Clocks
TxR_INSERTS	0x02	0-3	1	Egress Allocations
TxR_ADS_USED	0x04	0-3	1	
RING_BOUNCES	0x05	0-3	1	Number of LLC responses that bounced on the Ring.
RING_SRC_THRTL	0x07	0-3	1	Number of cycles the Cbo is actively throttling traffic onto the Ring in order to limit bounce traffic.
FAST_ASSERTED	0x09	0-1	1	FaST wire asserted
BOUNCE_CONTROL	0x0a	0-3	1	Bounce Control
RxR_OCCUPANCY	0x11	0	20	Ingress Occupancy
RxR_EXT_STARVED	0x12	0-3	1	Ingress Arbiter Blocking Cycles
RxR_INSERTS	0x13	0-3	1	Ingress Allocations
RING_AD_USED	0x1b	0-3	2	AD Ring In Use
RING_AK_USED	0x1c	0-3	2	AK Ring In Use
RING_BL_USED	0x1d	0-3	2	BL Ring in Use
RING_IV_USED	0x1e	0-3	1	BL Ring in Use
COUNTER0_OCCUPANCY	0x1f	0-3	20	Counter 0 Occupancy
RxR_IPQ_RETRY2	0x28	0-3	1	Probe Queue Retries
RxR_IRQ_RETRY2	0x29	0-3	1	Ingress Request Queue Rejects
RxR_ISMQ_RETRY2	0x2a	0-3	1	ISMQ Request Queue Rejects
RxR_IPQ_RETRY	0x31	0-3	1	Probe Queue Retries
RxR_IRQ_RETRY	0x32	0-3	1	Ingress Request Queue Rejects

Symbol Name	Event Code	Ctrs	Max Inc/Cyc	Description
RxR_ISMQ_RETRY	0x33	0-3	1	ISMQ Retries
LLC_LOOKUP	0x34	0-3	1	Cache Lookups
TOR_INSERTS	0x35	0-3	1	TOR Inserts
TOR_OCCUPANCY	0x36	0	20	TOR Occupancy
LLC_VICTIMS	0x37	0-3	1	Lines Victimized
MISC	0x39	0-3	1	Cbo Misc

2.3.5 CBO Box Common Metrics (Derived Events)

The following table summarizes metrics commonly calculated from CBO Box events.

Symbol Name: Definition	Equation
AVG_INGRESS_DEPTH: Average Depth of the Ingress Queue through the sample interval	RxR_OCCUPANCY.IRQ / SAMPLE_INTERVAL
AVG_INGRESS_LATENCY: Average Latency of Requests through the Ingress Queue in Uncore Clocks	RxR_OCCUPANCY.IRQ / RxR_INSERTS.IRQ
AVG_INGRESS_LATENCY_WHEN_NE: Average Latency of Requests through the Ingress Queue in Uncore Clocks when Ingress Queue has at least one entry	RxR_OCCUPANCY.IRQ / COUNTER0_OCCUPANCY{edge_det,thresh=0x1}
AVG_TOR_DRDS_MISS_WHEN_NE: Average Number of Data Read Entries that Miss the LLC when the TOR is not empty.	(TOR_OCCUPANCY.MISS_OPCODE / COUNTER0_OCCUPANCY{edge_det,thresh=0x1})) with:Cn_MSR_PMON_BOX_FILTER1.opc=0x182
AVG_TOR_DRDS_WHEN_NE: Average Number of Data Read Entries when the TOR is not empty.	(TOR_OCCUPANCY.OPCODE / COUNTER0_OCCUPANCY{edge_det,thresh=0x1}) with:Cn_MSR_PMON_BOX_FILTER1.opc=0x182
AVG_TOR_DRD_HIT_LATENCY: Average Latency of Data Reads through the TOR that hit the LLC	((TOR_OCCUPANCY.OPCODE - TOR_OCCUPANCY.MISS_OPCODE) / (TOR_INSERTS.OPCODE - TOR_INSERTS.MISS_OPCODE)) with:Cn_MSR_PMON_BOX_FILTER.opc=0x182
AVG_TOR_DRD_LATENCY: Average Latency of Data Read Entries making their way through the TOR	(TOR_OCCUPANCY.OPCODE / TOR_INSERTS.OPCODE) with:Cn_MSR_PMON_BOX_FILTER1.opc=0x182
AVG_TOR_DRD_LOC_MISS_LATENCY: Average Latency of Data Reads through the TOR that miss the LLC and were satsified by Local Memory	(TOR_OCCUPANCY.MISS_OPCODE / TOR_INSERTS.MISS_OPCODE) with:Cn_MSR_PMON_BOX_FILTER1.{opc,nid}={0x182,my_node}
AVG_TOR_DRD_MISS_LATENCY: Average Latency of Data Reads through the TOR that miss the LLC	(TOR_OCCUPANCY.MISS_OPCODE / TOR_INSERTS.MISS_OPCODE) with:Cn_MSR_PMON_BOX_FILTER1.opc=0x182
AVG_TOR_DRD_REM_MISS_LATENCY: Average Latency of Data Reads through the TOR that miss the LLC and were satsified by a Remote cache or Remote Memory	(TOR_OCCUPANCY.MISS_OPCODE / TOR_INSERTS.MISS_OPCODE) with:Cn_MSR_PMON_BOX_FILTER.{opc,nid}={0x182,other_nodes}
CYC_INGRESS_BLOCKED: Cycles the Ingress Request Queue arbiter was Blocked	RxR_EXT_STARVED.IRQ / SAMPLE_INTERVAL
CYC_USED_DN: Cycles Used in the Down direction, Even polarity	RING_BL_USED.CCW / SAMPLE_INTERVAL
CYC_USED_UP: Cycles Used in the Up direction, Even polarity	RING_BL_USED.CW / SAMPLE_INTERVAL

Symbol Name: Definition	Equation
FAST_STR_LLC_MISS: Number of ItoM (fast string) operations that miss the LLC	TOR_INSERTS.MISS_OPCODE with:Cn_MSR_PMON_BOX_FILTER1.opc=0x1C8
FAST_STR_LLC_REQ: Number of ItoM (fast string) operations that reference the LLC	TOR_INSERTS.OPCODE with:Cn_MSR_PMON_BOX_FILTER1.opc=0x1C8
INGRESS_REJ_V_INS: Ratio of Ingress Request Entries that were rejected vs. inserted	RxR_INSERTS.IRQ_REJ / RxR_INSERTS.IRQ
IO_READ_BW: IO Read Bandwidth in MB - Disk or Network Reads	(TOR_INSERTS.OPCODE with:{Cn_MSR_PMON_BOX_FILTER0.tid=0x3F, Cn_MSR_PMON_BOX_FILTER1.opc=0x1C8} + TOR_INSERTS.OPCODE with:Cn_MSR_PMON_BOX_FILTER.opc=0x1E6) * 64 / 1000000
IO_WRITE_BW: IO Write Bandwidth in MB - Disk or Network Writes	(TOR_INSERTS.OPCODE with:Cn_MSR_PMON_BOX_FILTER1.opc=0x19E + TOR_INSERTS.OPCODE with:Cn_MSR_PMON_BOX_FILTER.opc=0x1E4) * 64 / 1000000
LLC_DRD_MISS_PCT: LLC Data Read miss ratio	LLC_LOOKUP.DATA_READ with:Cn_MSR_PMON_BOX_FILTER0.state=0x1 / LLC_LOOKUP.DATA_READ with:Cn_MSR_PMON_BOX_FILTER.state=0x3F
LLC_DRD_RFO_MISS_TO_LOC_MEM: LLC Data Read and RFO misses satisfied by local memory.	(TOR_INSERTS.NID_MISS_OPCODE with:Cn_MSR_PMON_BOX_FILTER1.{opc,nid}={0x182,my_node} + TOR_INSERTS.NID_MISS_OPCODE with:Cn_MSR_PMON_BOX_FILTER.{opc,nid}={0x180,my_node}) / (TOR_INSERTS.NID_MISS_OPCODE with:Cn_MSR_PMON_BOX_FILTER.{opc,nid}={0x182,0xF} + TOR_INSERTS.NID_MISS_OPCODE with:Cn_MSR_PMON_BOX_FILTER.{opc,nid}={0x180,0xF})
LLC_DRD_RFO_MISS_TO_REM_MEM: LLC Data Read and RFO misses satisfied by a remote cache or remote memory.	(TOR_INSERTS.NID_MISS_OPCODE with:Cn_MSR_PMON_BOX_FILTER1.{opc,nid}={0x182,other_nodes} + TOR_INSERTS.NID_MISS_OPCODE with:Cn_MSR_PMON_BOX_FILTER.{opc,nid}={0x180,other_nodes}) / (TOR_INSERTS.NID_MISS_OPCODE with:Cn_MSR_PMON_BOX_FILTER.{opc,nid}={0x182,0xF} + TOR_INSERTS.NID_MISS_OPCODE with:Cn_MSR_PMON_BOX_FILTER.{opc,nid}={0x180,0xF})
LLC_MPI: LLC Misses Per Instruction (code, read, RFO and prefetches)	LLC_LOOKUP.ANY (Cn_MSR_PMON_BOX_FILTER0.state=0x1) / INST_RETIRED.ALL (on Core)
LLC_PCIE_DATA_BYTES: LLC write miss (disk/network reads) bandwidth in MB	TOR_INSERTS.OPCODE with:{Cn_MSR_PMON_BOX_FILTER0.tid=0x3F, Cn_MSR_PMON_BOX_FILTER1.opc=0x1C8} * 64
LLC_RFO_MISS_PCT: LLC RFO Miss Ratio	(TOR_INSERTS.MISS_OPCODE / TOR_INSERTS.OPCODE) with:Cn_MSR_PMON_BOX_FILTER1.opc=0x180
MEM_WB_BYTES: Data written back to memory in Number of Bytes	LLC_VICTIMS.M_STATE * 64
PARTIAL_PCI_READS: Number of partial PCI reads	TOR_INSERTS.OPCODE with:{Cn_MSR_PMON_BOX_FILTER0.tid=0x3F, Cn_MSR_PMON_BOX_FILTER1.opc=0x187}
PARTIAL_PCI_WRITES: Number of partial PCI writes	TOR_INSERTS.OPCODE with:Cn_MSR_PMON_BOX_FILTER1.opc=0x1E5
PCIE_DATA_BYTES: Data from PCIe in Number of Bytes	(TOR_INSERTS.OPCODE with:Cn_MSR_PMON_BOX_FILTER1.opc=0x194 + TOR_INSERTS.OPCODE with:{Cn_MSR_PMON_BOX_FILTER0.tid=0x3F, Cn_MSR_PMON_BOX_FILTER1.opc=0x1C8}) * 64
RING_THRU_DN_BYTES: Ring throughput in the Down direction, Even polarity in Bytes	RING_BL_USED.CCW* 32

Symbol Name: Definition	Equation
RING_THRU_UP_BYTES: Ring throughput in the Up direction, Even polarity in Bytes	RING_BL_USED.CW * 32
STREAMED_FULL_STORES: Number of Streamed Store (of Full Cache Line) Transactions	TOR_INSERTS.OPCODE with:Cn_MSR_PMON_BOX_FILTER1.opc=0x18C
STREAMED_PART_STORES: Number of Streamed Store (of Partial Cache Line) Transactions	TOR_INSERTS.OPCODE with:Cn_MSR_PMON_BOX_FILTER1.opc=0x18D
UC_READS: Uncachable Read Transactions	TOR_INSERTS.MISS_OPCODE with:Cn_MSR_PMON_BOX_FILTER1.opc=0x187

2.3.6 CBO Box Performance Monitor Event List

The section enumerates Intel® Xeon® Processor D-1500 Product Family performance monitoring events for the CBO Box.

BOUNCE_CONTROL

- **Title:** Bounce Control
- **Category:** RING Events
- **Event Code:** 0x0a
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

CLOCKTICKS

- **Title:** Uncore Clocks
- **Category:** UCLK Events
- **Event Code:** 0x00
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

COUNTER0_OCCUPANCY

- **Title:** Counter 0 Occupancy
- **Category:** OCCUPANCY Events
- **Event Code:** 0x1f
- **Max. Inc/Cyc:.** 20, **Register Restrictions:** 0-3
- **Definition:** Since occupancy counts can only be captured in the Cbo's 0 counter, this event allows a user to capture occupancy related information by filtering the Cb0 occupancy count captured in Counter 0. The filtering available is found in the control register - threshold, invert and edge detect. E.g. setting threshold to 1 can effectively monitor how many cycles the monitored queue has an entry.

FAST_ASSERTED

- **Title:** FaST wire asserted
- **Category:** EGRESS Events
- **Event Code:** 0x09
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** Counts the number of cycles either the local distress or incoming distress signals are asserted. Incoming distress includes both up and dn.

LLC_LOOKUP

- **Title:** Cache Lookups
- **Category:** CACHE Events
- **Event Code:** 0x34
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times the LLC was accessed - this includes code, data, prefetches and hints coming from L2. This has numerous filters available. Note the non-standard filtering equation. This event will count requests that lookup the cache multiple times with multiple increments. One must ALWAYS set umask bit 0 and select a state or states to match. Otherwise, the event will count nothing. CBoGlCtrl[22:18] bits correspond to [FMESI] state.
- **NOTE:** Bit 0 of the umask must always be set for this event. This allows us to match a given state (or states). The state is programmed in Cn_MSR_PMON_BOX_FILTER.state. The state field is a bit mask, so you can select (and monitor) multiple states at a time. 0 = I (miss), 1 = S, 2 = E, 3 = M, 4 = F. For example, if you wanted to monitor F and S hits, you could set 10010b in the 5-bit state field. To monitor any lookup, set the field to 0x1F.

Table 2-21. Unit Masks for LLC_LOOKUP

Extension	umask [15:8]	Filter Dep	Description
DATA_READ	b00000011	CBoFilter0[23:17]	Data Read Request Read transactions
WRITE	b00000101	CBoFilter0[23:17]	Write Requests Writeback transactions from L2 to the LLC This includes all write transactions -- both Cachable and UC.
REMOTE_SNOOP	b00001001	CBoFilter0[23:17]	External Snoop Request Filters for only snoop requests coming from the remote socket(s) through the IPQ.
ANY	b00010001	CBoFilter0[23:17]	Any Request Filters for any transaction originating from the IPQ or IRQ. This does not include lookups originating from the ISMQ.
READ	b00100001	CBoFilter0[22:18]	Any Read Request Read transactions
NID	b01000001	CBoFilter0[23:17]	Lookups that Match NID Qualify one of the other subevents by the Target NID. The NID is programmed in Cn_MSR_PMON_BOX_FILTER.nid. In conjunction with STATE = I, it is possible to monitor misses to specific NIDs in the system.

LLC_VICTIMS

- **Title:** Lines Victimized
- **Category:** CACHE Events
- **Event Code:** 0x37
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of lines that were victimized on a fill. This can be filtered by the state that the line was in.

Table 2-22. Unit Masks for LLC_VICTIMS

Extension	umask [15:8]	Filter Dep	Description
M_STATE	bxxxxxxx1		Lines in M state
E_STATE	bxxxxxx1x		Lines in E state
S_STATE	bxxxxx1xx		Lines in S State

Table 2-22. Unit Masks for LLC_VICTIMS

Extension	umask [15:8]	Filter Dep	Description
F_STATE	bxxxx1xxx		
MISS	bxxx1xxxx		
NID	bx1xxxxxx	CBoFilter1[17:10]	Victimized Lines that Match NID Qualify one of the other subevents by the Target NID. The NID is programmed in Cn_MSR_PMON_BOX_FILTER.nid. In conjunction with STATE = I, it is possible to monitor misses to specific NIDs in the system.

MISC

- **Title:** Cbo Misc
- **Category:** MISC Events
- **Event Code:** 0x39
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Miscellaneous events in the Cbo.

Table 2-23. Unit Masks for MISC

Extension	umask [15:8]	Description
RSPI_WAS_FSE	bxxxxxxx1	Silent Snoop Eviction Counts the number of times when a Snoop hit in FSE states and triggered a silent eviction. This is useful because this information is lost in the PRE encodings.
WC_ALIASING	bxxxxxx1x	Write Combining Aliasing Counts the number of times that a USWC write (WCIL(F)) transaction hit in the LLC in M state, triggering a WBMtoI followed by the USWC write. This occurs when there is WC aliasing.
STARTED	bxxxxx1xx	
RFO_HIT_S	bxxxx1xxx	RFO HitS Number of times that an RFO hit in S state. This is useful for determining if it might be good for a workload to use RspIWB instead of RspSWB.
CVZERO_PREFETCH_VICTIM	bxxx1xxxx	Clean Victim with raw CV=0
CVZERO_PREFETCH_MISS	bxx1xxxxx	DRd hitting non-M with raw CV=0

RING_AD_USED

- **Title:** AD Ring In Use
- **Category:** RING Events
- **Event Code:** 0x1b
- **Max. Inc/Cyc:.** 2, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles that the AD ring is being used at this ring stop. This includes when packets are passing by and when packets are being sunk, but does not include when packets are being sent from the ring stop. There are two rings -- a clockwise ring and a counter-clockwise ring. On the left side of the ring, the "UP" direction is on the clockwise ring and "DN" is on the counter-clockwise ring. On the right side of the ring, this is reversed. The first half of the CBos are on the left side of the ring, and the 2nd half are on the right side of the ring. In other words (for example), in a 4c part, Cbo 0 UP AD is NOT the same ring as CBo 2 UP AD because they are on opposite sides of the ring.
- **NOTE:** In any cycle, a ring stop can see up to one packet moving in the UP direction and one packet moving in the DN direction.

Table 2-24. Unit Masks for RING_AD_USED

Extension	umask [15:8]	Description
UP_EVEN	bxxxxxxx1	Up and Even Filters for the Up and Even ring polarity.
UP_ODD	bxxxxxx1x	Up and Odd Filters for the Up and Odd ring polarity.
UP	b00000011	Up
DOWN_EVEN	bxxxxx1xx	Down and Even Filters for the Down and Even ring polarity.
DOWN_ODD	bxxxx1xxx	Down and Odd Filters for the Down and Odd ring polarity.
DOWN	b00001100	Down
ALL	b00001111	All

RING_AK_USED

- **Title:** AK Ring In Use
- **Category:** RING Events
- **Event Code:** 0x1c
- **Max. Inc/Cyc:.** 2, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles that the AK ring is being used at this ring stop. This includes when packets are passing by and when packets are being sunk, but does not include when packets are being sent from the ring stop. There are two rings -- a clockwise ring and a counter-clockwise ring. On the left side of the ring, the "UP" direction is on the clockwise ring and "DN" is on the counter-clockwise ring. On the right side of the ring, this is reversed. The first half of the CBos are on the left side of the ring, and the 2nd half are on the right side of the ring. In other words (for example), in a 4c part, Cbo 0 UP AD is NOT the same ring as CBo 2 UP AD because they are on opposite sides of the ring.
- **NOTE:** In any cycle, a ring stop can see up to one packet moving in the UP direction and one packet moving in the DN direction.

Table 2-25. Unit Masks for RING_AK_USED

Extension	umask [15:8]	Description
UP_EVEN	bxxxxxxx1	Up and Even Filters for the Up and Even ring polarity.
UP_ODD	bxxxxxx1x	Up and Odd Filters for the Up and Odd ring polarity.
UP	b00000011	Up
DOWN_EVEN	bxxxxx1xx	Down and Even Filters for the Down and Even ring polarity.
DOWN_ODD	bxxxx1xxx	Down and Odd Filters for the Down and Odd ring polarity.
DOWN	b00001100	Down
ALL	b00001111	All

RING_BL_USED

- **Title:** BL Ring in Use
- **Category:** RING Events
- **Event Code:** 0x1d
- **Max. Inc/Cyc:.** 2, **Register Restrictions:** 0-3

- **Definition:** Counts the number of cycles that the BL ring is being used at this ring stop. This includes when packets are passing by and when packets are being sunk, but does not include when packets are being sent from the ring stop. There are two rings -- a clockwise ring and a counter-clockwise ring. On the left side of the ring, the "UP" direction is on the clockwise ring and "DN" is on the counter-clockwise ring. On the right side of the ring, this is reversed. The first half of the CBos are on the left side of the ring, and the 2nd half are on the right side of the ring. In other words (for example), in a 4c part, Cbo 0 UP AD is NOT the same ring as CBo 2 UP AD because they are on opposite sides of the ring.
- **NOTE:** In any cycle, a ring stop can see up to one packet moving in the UP direction and one packet moving in the DN direction.

Table 2-26. Unit Masks for RING_BL_USED

Extension	umask [15:8]	Description
UP_EVEN	bxxxxxxx1	Up and Even Filters for the Up and Even ring polarity.
UP_ODD	bxxxxxx1x	Up and Odd Filters for the Up and Odd ring polarity.
UP	b00000011	Up
DOWN_EVEN	bxxxxx1xx	Down and Even Filters for the Down and Even ring polarity.
DOWN_ODD	bxxxx1xxx	Down and Odd Filters for the Down and Odd ring polarity.
DOWN	b00001100	Down
ALL	b00001111	Down

RING_BOUNCES

- **Title:** Number of LLC responses that bounced on the Ring.
- **Category:** RING Events
- **Event Code:** 0x05
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-27. Unit Masks for RING_BOUNCES

Extension	umask [15:8]	Description
AD	bxxxxxxx1	AD
AK	bxxxxxx1x	AK
BL	bxxxxx1xx	BL
IV	bxxx1xxxx	Snoops of processor's cache.

RING_IV_USED

- **Title:** BL Ring in Use
- **Category:** RING Events
- **Event Code:** 0x1e
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles that the IV ring is being used at this ring stop. This includes when packets are passing by and when packets are being sunk, but does not include when packets are being sent from the ring stop. There is only 1 IV ring. Therefore, if one wants to monitor the "Even" ring, they should select both UP_EVEN and DN_EVEN. To monitor the "Odd" ring, they should select both UP_ODD and DN_ODD.
- **NOTE:** IV messages are split into two parts. In any cycle, a ring stop can see up to one (half-)packet moving in the UP direction and one (half-)packet moving in the DN direction.

Table 2-28. Unit Masks for RING_IV_USED

Extension	umask [15:8]	Description
UP	b00000011	Filters any polarity
DN	b00001100	Filters any polarity
ANY	b00001111	Any Filters any polarity
DOWN	b11001100	Down Filters for Down polarity

RING_SRC_THRTL

- **Title:** Number of cycles the Cbo is actively throttling traffic onto the Ring in order to limit bounce traffic.
- **Category:** RING Events
- **Event Code:** 0x07
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

RxR_EXT_STARVED

- **Title:** Ingress Arbiter Blocking Cycles
- **Category:** INGRESS Events
- **Event Code:** 0x12
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts cycles in external starvation. This occurs when one of the ingress queues is being starved by the other queues.

Table 2-29. Unit Masks for RxR_EXT_STARVED

Extension	umask [15:8]	Description
IRQ	bxxxxxxx1	IPQ IRQ is externally starved and therefore we are blocking the IPQ.
IPQ	bxxxxxx1x	IRQ IPQ is externally startved and therefore we are blocking the IRQ.
PRQ	bxxxxx1xx	PRQ
ISMQ_BIDS	bxxxx1xxx	ISMQ_BID Number of times that the ISMQ Bid.

RxR_INSERTS

- **Title:** Ingress Allocations
- **Category:** INGRESS Events
- **Event Code:** 0x13
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts number of allocations per cycle into the specified Ingress queue.
- **NOTE:** IRQ_REJECTED should not be Ored with the other umasks.

Table 2-30. Unit Masks for RxR_INSERTS

Extension	umask [15:8]	Description
IRQ	bxxxxxxx1	IRQ
IRQ_REJ	bxxxxxx1x	IRQ Rejected
IPQ	bxxxxx1xx	IPQ
PRQ	bxxx1xxxx	PRQ
PRQ_REJ	bxx1xxxxx	PRQ

RxR_IPQ_RETRY

- **Title:** Probe Queue Retries
- **Category:** INGRESS_RETRY Events
- **Event Code:** 0x31
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of times a snoop (probe) request had to retry. Filters exist to cover some of the common cases retries.

Table 2-31. Unit Masks for RxR_IPQ_RETRY

Extension	umask [15:8]	Description
ANY	bxxxxxxx1	Any Reject Counts the number of times that a request form the IPQ was retried because of a TOR reject. TOR rejects from the IPQ can be caused by the Egress being full or Address Conflicts.
FULL	bxxxxxx1x	No Egress Credits Counts the number of times that a request form the IPQ was retried because of a TOR reject from the Egress being full. IPQ requests make use of the AD Egress for regular responses, the BL egress to forward data, and the AK egress to return credits.
ADDR_CONFLICT	bxxxxx1xx	Address Conflict Counts the number of times that a request form the IPQ was retried because of a TOR reject from an address conflicts. Address conflicts out of the IPQ should be rare. They will generally only occur if two different sockets are sending requests to the same address at the same time. This is a true "conflict" case, unlike the IPQ Address Conflict which is commonly caused by prefetching characteristics.
QPI_CREDITS	bxxx1xxxx	No Intel QPI Credits

RxR_IPQ_RETRY2

- **Title:** Probe Queue Retries
- **Category:** INGRESS_RETRY Events
- **Event Code:** 0x28
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of times a snoop (probe) request had to retry. Filters exist to cover some of the common cases retries.

Table 2-32. Unit Masks for RxR_IPQ_RETRY2

Extension	umask [15:8]	Filter Dep	Description
AD_SBO	bxxxxxxx1		No AD Sbo Credits Counts the number of times that a request from the IPQ was retried because of it lacked credits to send an AD packet to the Sbo.
TARGET	bx1xxxxxx	CBoFilter1[15:0]	Target Node Filter Counts the number of times that a request from the IPQ was retried filtered by the Target NodeID as specified in the Cbox's Filter register.

RxR_IRQ_RETRY

- **Title:** Ingress Request Queue Rejects
- **Category:** INGRESS_RETRY Events
- **Event Code:** 0x32
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-33. Unit Masks for RxR_IRQ_RETRY

Extension	umask [15:8]	Filter Dep	Description
ANY	bxxxxxxx1		Any Reject Counts the number of IRQ retries that occur. Requests from the IRQ are retried if they are rejected from the TOR pipeline for a variety of reasons. Some of the most common reasons include if the Egress is full, there are no RTIDs, or there is a Physical Address match to another outstanding request.
FULL	bxxxxxx1x		No Egress Credits Counts the number of times that a request from the IRQ was retried because it failed to acquire an entry in the Egress. The egress is the buffer that queues up for allocating onto the ring. IRQ requests can make use of all four rings and all four Egresses. If any of the queues that a given request needs to make use of are full, the request will be retried.
ADDR_CONFLICT	bxxxxx1xx		Address Conflict Counts the number of times that a request from the IRQ was retried because of an address match in the TOR. In order to maintain coherency, requests to the same address are not allowed to pass each other up in the Cbo. Therefore, if there is an outstanding request to a given address, one cannot issue another request to that address until it is complete. This comes up most commonly with prefetches. Outstanding prefetches occasionally will not complete their memory fetch and a demand request to the same address will then sit in the IRQ and get retried until the prefetch fills the data into the LLC. Therefore, it will not be uncommon to see this case in high bandwidth streaming workloads when the LLC Prefetcher in the core is enabled.

Table 2-33. Unit Masks for RxR_IRQ_RETRY

Extension	umask [15:8]	Filter Dep	Description
RTID	bxxxx1xxx		No RTIDs Counts the number of times that requests from the IRQ were retried because there were no RTIDs available. RTIDs are required after a request misses the LLC and needs to send snoops and/or requests to memory. If there are no RTIDs available, requests will queue up in the IRQ and retry until one becomes available. Note that there are multiple RTID pools for the different sockets. There may be cases where the local RTIDs are all used, but requests destined for remote memory can still acquire an RTID because there are remote RTIDs available. This event does not provide any filtering for this case.
QPI_CREDITS	bxxx1xxxx		No Intel QPI Credits Number of requests rejects because of lack of Intel QPI Ingress credits. These credits are required in order to send transactions to the Intel QPI agent. Please see the QPI_IGR_CREDITS events for more information.
IIO_CREDITS	bxx1xxxxx		No IIO Credits Number of times a request attempted to acquire the NCS/NCB credit for sending messages on BL to the IIO. There is a single credit in each CBo that is shared between the NCS and NCB message classes for sending transactions on the BL ring (such as read data) to the IIO.
NID	bx1xxxxxx	CBoFilter1[15:0]	Qualify one of the other subevents by a given RTID destination NID. The NID is programmed in Cn_MSR_PMON_BOX_FILTER1.nid.

RxR_IRQ_RETRY2

- **Title:** Ingress Request Queue Rejects
- **Category:** INGRESS_RETRY Events
- **Event Code:** 0x29
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-34. Unit Masks for RxR_IRQ_RETRY2

Extension	umask [15:8]	Filter Dep	Description
AD_SBO	bxxxxxxx1		No AD Sbo Credits Counts the number of times that a request from the IPQ was retried because of it lacked credits to send an AD packet to the Sbo.
BL_SBO	bxxxxxx1x		No BL Sbo Credits Counts the number of times that a request from the IPQ was retried because of it lacked credits to send an BL packet to the Sbo.
TARGET	bx1xxxxxx	CBoFilter1[15:0]	Target Node Filter Counts the number of times that a request from the IPQ was retried filtered by the Target NodeID as specified in the Cbox's Filter register.

RxR_ISMQ_RETRY

- **Title:** ISMQ Retries
- **Category:** INGRESS_RETRY Events
- **Event Code:** 0x33
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3

- **Definition:** Number of times a transaction flowing through the ISMQ had to retry. Transaction pass through the ISMQ as responses for requests that already exist in the Cbo. Some examples include: when data is returned or when snoop responses come back from the cores.

Table 2-35. Unit Masks for RxR_ISMQ_RETRY

Extension	umask [15:8]	Filter Dep	Description
ANY	bxxxxxxx1		Any Reject Counts the total number of times that a request from the ISMQ retried because of a TOR reject. ISMQ requests generally will not need to retry (or at least ISMQ retries are less common than IRQ retries). ISMQ requests will retry if they are not able to acquire a needed Egress credit to get onto the ring, or for cache evictions that need to acquire an RTID. Most ISMQ requests already have an RTID, so eviction retries will be less common here.
FULL	bxxxxxx1x		No Egress Credits Counts the number of times that a request from the ISMQ retried because of a TOR reject caused by a lack of Egress credits. The egress is the buffer that queues up for allocating onto the ring. If any of the Egress queues that a given request needs to make use of are full, the request will be retried.
RTID	bxxxx1xxx		No RTIDs Counts the number of times that a request from the ISMQ retried because of a TOR reject caused by no RTIDs. M-state cache evictions are serviced through the ISMQ, and must acquire an RTID in order to write back to memory. If no RTIDs are available, they will be retried.
QPI_CREDITS	bxxx1xxxx		No Intel QPI Credits
IIO_CREDITS	bxx1xxxxx		No IIO Credits Number of times a request attempted to acquire the NCS/NCB credit for sending messages on BL to the IIO. There is a single credit in each CBo that is shared between the NCS and NCB message classes for sending transactions on the BL ring (such as read data) to the IIO.
NID	bx1xxxxxx	CBoFilter1[15:0]	Qualify one of the other subevents by a given RTID destination NID. The NID is programmed in Cn_MSR_PMON_BOX_FILTER1.nid.
WB_CREDITS	b1xxxxxxx	CBoFilter1[15:0]	Qualify one of the other subevents by a given RTID destination NID. The NID is programmed in Cn_MSR_PMON_BOX_FILTER1.nid.

RxR_ISMQ_RETRY2

- **Title:** ISMQ Request Queue Rejects
- **Category:** INGRESS_RETRY Events
- **Event Code:** 0x2a
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-36. Unit Masks for RxR_ISMQ_RETRY2

Extension	umask [15:8]	Filter Dep	Description
AD_SBO	bxxxxxxx1		No AD Sbo Credits Counts the number of times that a request from the ISMQ was retried because of it lacked credits to send an AD packet to the Sbo.
BL_SBO	bxxxxxx1x		No BL Sbo Credits Counts the number of times that a request from the ISMQ was retried because of it lacked credits to send an BL packet to the Sbo.
TARGET	bx1xxxxxx	CBoFilter1[15:0]	Target Node Filter Counts the number of times that a request from the ISMQ was retried filtered by the Target NodeID as specified in the Cbox's Filter register.

RxR_OCCUPANCY

- **Title:** Ingress Occupancy
- **Category:** INGRESS Events
- **Event Code:** 0x11
- **Max. Inc/Cyc:.** 20, **Register Restrictions:** 0
- **Definition:** Counts number of entries in the specified Ingress queue in each cycle.
- **NOTE:** IRQ_REJECTED should not be Ored with the other umasks.

Table 2-37. Unit Masks for RxR_OCCUPANCY

Extension	umask [15:8]	Description
IRQ	b00000001	IRQ
IRQ_REJ	b00000010	IRQ Rejected
IPQ	b00000100	IPQ
PRQ_REJ	b00100000	PRQ Rejects

TOR_INSERTS

- **Title:** TOR Inserts
- **Category:** TOR Events
- **Event Code:** 0x35
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of entries successfulynserted into the TOR that match qualifications specified by the subevent. There are a number of subevent 'filters' but only a subset of the subevent combinations are valid. Subevents that require an opcode or NID match require the Cn_MSR_PMON_BOX_FILTER.{opc, nid} field to be set. If, for example, one wanted to count DRD Local Misses, one should select "MISS_OPC_MATCH" and set Cn_MSR_PMON_BOX_FILTER.opc to DRD (0x182).

Table 2-38. Unit Masks for TOR_INSERTS

Extension	umask [15:8]	Filter Dep	Description
OPCODE	b00000001	CBoFilter1[28:20]	Opcode Match Transactions inserted into the TOR that match an opcode (matched by Cn_MSR_PMON_BOX_FILTER.opc)
MISS_OPCODE	b00000011	CBoFilter1[28:20]	Miss Opcode Match Miss transactions inserted into the TOR that match an opcode.
EVICTION	b00000100		Evictions Eviction transactions inserted into the TOR. Evictions can be quick, such as when the line is in the F, S, or E states and no core valid bits are set. They can also be longer if either CV bits are set (so the cores need to be snooped) and/or if there is a HitM (in which case it is necessary to write the request out to memory).
ALL	b00001000		All All transactions inserted into the TOR. This includes requests that reside in the TOR for a short time, such as LLC Hits that do not need to snoop cores or requests that get rejected and have to be retried through one of the ingress queues. The TOR is more commonly a bottleneck in skews with smaller core counts, where the ratio of RTIDs to TOR entries is larger. Note that there are reserved TOR entries for various request types, so it is possible that a given request type be blocked with an occupancy that is less than 20. Also note that generally requests will not be able to arbitrate into the TOR pipeline if there are no available TOR slots.
WB	b00010000		Writebacks Write transactions inserted into the TOR. This does not include "RFO", but actual operations that contain data being sent from the core.
LOCAL_OPCODE	b00100001	CBoFilter1[28:20]	Local Memory - Opcode Matched All transactions, satisfied by an opcode, inserted into the TOR that are satisfied by locally HOMed memory.
MISS_LOCAL_OPCODE	b00100011	CBoFilter1[28:20]	Misses to Local Memory - Opcode Matched Miss transactions, satisfied by an opcode, inserted into the TOR that are satisfied by locally HOMed memory.
LOCAL	b00101000		Local Memory All transactions inserted into the TOR that are satisfied by locally HOMed memory.
MISS_LOCAL	b00101010		Misses to Local Memory Miss transactions inserted into the TOR that are satisfied by locally HOMed memory.
NID_OPCODE	b01000001	CBoFilter1[28:20], CBoFilter1[15:0]	NID and Opcode Matched Transactions inserted into the TOR that match a NID and an opcode.
NID_MISS_OPCODE	b01000011	CBoFilter1[28:20], CBoFilter1[15:0]	NID and Opcode Matched Miss Miss transactions inserted into the TOR that match a NID and an opcode.
NID_EVICTION	b01000100	CBoFilter1[15:0]	NID Matched Evictions NID matched eviction transactions inserted into the TOR.
NID_ALL	b01001000	CBoFilter1[15:0]	NID Matched All NID matched (matches an RTID destination) transactions inserted into the TOR. The NID is programmed in Cn_MSR_PMON_BOX_FILTER.nid. In conjunction with STATE = I, it is possible to monitor misses to specific NIDs in the system.

Table 2-38. Unit Masks for TOR_INSERTS

Extension	umask [15:8]	Filter Dep	Description
NID_MISS_ALL	b01001010	CBoFilter1[15:0]	NID Matched Miss All All NID matched miss requests that were inserted into the TOR.
NID_WB	b01010000	CBoFilter1[15:0]	NID Matched Writebacks NID matched write transactions inserted into the TOR.
REMOTE_OPCODE	b10000001	CBoFilter1[28:20]	Remote Memory - Opcode Matched All transactions, satisfied by an opcode, inserted into the TOR that are satisfied by remote caches or remote memory.
MISS_REMOTE_OPCODE	b10000011	CBoFilter1[28:20]	Misses to Remote Memory - Opcode Matched Miss transactions, satisfied by an opcode, inserted into the TOR that are satisfied by remote caches or remote memory.
REMOTE	b10001000		Remote Memory All transactions inserted into the TOR that are satisfied by remote caches or remote memory.
MISS_REMOTE	b10001010		Misses to Remote Memory Miss transactions inserted into the TOR that are satisfied by remote caches or remote memory.

TOR_OCCUPANCY

- **Title:** TOR Occupancy
- **Category:** TOR Events
- **Event Code:** 0x36
- **Max. Inc/Cyc:.** 20, **Register Restrictions:** 0
- **Definition:** For each cycle, this event accumulates the number of valid entries in the TOR that match qualifications specified by the subevent. There are a number of subevent 'filters' but only a subset of the subevent combinations are valid. Subevents that require an opcode or NID match require the Cn_MSR_PMON_BOX_FILTER.{opc, nid} field to be set. If, for example, one wanted to count DRD Local Misses, one should select "MISS_OPC_MATCH" and set Cn_MSR_PMON_BOX_FILTER.opc to DRD (0x182)

Table 2-39. Unit Masks for TOR_OCCUPANCY

Extension	umask [15:8]	Filter Dep	Description
OPCODE	b00000001	CBoFilter1[28:20]	Opcode Match TOR entries that match an opcode (matched by Cn_MSR_PMON_BOX_FILTER.opc).
MISS_OPCODE	b00000011	CBoFilter1[28:20]	Miss Opcode Match TOR entries for miss transactions that match an opcode. This generally means that the request was sent to memory or MMIO.
EVICTION	b00000100		Evictions Number of outstanding eviction transactions in the TOR. Evictions can be quick, such as when the line is in the F, S, or E states and no core valid bits are set. They can also be longer if either CV bits are set (so the cores need to be snooped) and/or if there is a HitM (in which case it is necessary to write the request out to memory).

Table 2-39. Unit Masks for TOR_OCCUPANCY

Extension	umask [15:8]	Filter Dep	Description
ALL	b00001000		Any All valid TOR entries. This includes requests that reside in the TOR for a short time, such as LLC Hits that do not need to snoop cores or requests that get rejected and have to be retried through one of the ingress queues. The TOR is more commonly a bottleneck in skews with smaller core counts, where the ratio of RTIDs to TOR entries is larger. Note that there are reserved TOR entries for various request types, so it is possible that a given request type be blocked with an occupancy that is less than 20. Also note that generally requests will not be able to arbitrate into the TOR pipeline if there are no available TOR slots.
MISS_ALL	b00001010		Miss All Number of outstanding miss requests in the TOR. 'Miss' means the allocation requires an RTID. This generally means that the request was sent to memory or MMIO.
WB	b00010000		Writebacks Write transactions in the TOR. This does not include "RFO", but actual operations that contain data being sent from the core.
LOCAL_OPCODE	b00100001	CBoFilter1[28:20]	Local Memory - Opcode Matched Number of outstanding transactions, satisfied by an opcode, in the TOR that are satisfied by locally HOMed memory.
MISS_LOCAL_OPCODE	b00100011	CBoFilter1[28:20]	Misses to Local Memory - Opcode Matched Number of outstanding Miss transactions, satisfied by an opcode, in the TOR that are satisfied by locally HOMed memory.
LOCAL	b00101000		
MISS_LOCAL	b00101010		
NID_OPCODE	b01000001	CBoFilter1[28:20], CBoFilter1[15:0]	NID and Opcode Matched TOR entries that match a NID and an opcode.
NID_MISS_OPCODE	b01000011	CBoFilter1[28:20], CBoFilter1[15:0]	NID and Opcode Matched Miss Number of outstanding Miss requests in the TOR that match a NID and an opcode.
NID_EVICTION	b01000100	CBoFilter1[15:0]	NID Matched Evictions Number of outstanding NID matched eviction transactions in the TOR .
NID_ALL	b01001000	CBoFilter1[15:0]	NID Matched Number of NID matched outstanding requests in the TOR. The NID is programmed in Cn_MSR_PMON_BOX_FILTER.nid.In conjunction with STATE = I, it is possible to monitor misses to specific NIDs in the system.
NID_MISS_ALL	b01001010	CBoFilter1[15:0]	NID Matched Number of outstanding Miss requests in the TOR that match a NID.
NID_WB	b01010000	CBoFilter1[15:0]	NID Matched Writebacks NID matched write transactions int the TOR.
REMOTE_OPCODE	b10000001	CBoFilter1[28:20]	Remote Memory - Opcode Matched Number of outstanding transactions, satisfied by an opcode, in the TOR that are satisfied by remote caches or remote memory.

Table 2-39. Unit Masks for TOR_OCCUPANCY

Extension	umask [15:8]	Filter Dep	Description
MISS_REMOTE_OPCODE	b10000011	CBoFilter1[28:20]	Misses to Remote Memory - Opcode Matched Number of outstanding Miss transactions, satisfied by an opcode, in the TOR that are satisfied by remote caches or remote memory.
REMOTE	b10001000		
MISS_REMOTE	b10001010		

TxR_ADS_USED

- **Title:**
- **Category:** EGRESS Events
- **Event Code:** 0x04
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-40. Unit Masks for TxR_ADS_USED

Extension	umask [15:8]	Description
AD	bxxxxxxx1	Onto AD Ring
AK	bxxxxxx1x	Onto AK Ring
BL	bxxxxx1xx	Onto BL Ring

TxR_INSERTS

- **Title:** Egress Allocations
- **Category:** EGRESS Events
- **Event Code:** 0x02
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of allocations into the Cbo Egress. The Egress is used to queue up requests destined for the ring.

Table 2-41. Unit Masks for TxR_INSERTS

Extension	umask [15:8]	Description
AD_CACHE	bxxxxxxx1	AD - Cachebo Ring transactions from the Cachebo destined for the AD ring. Some example include outbound requests, snoop requests, and snoop responses.
AK_CACHE	bxxxxxx1x	AK - Cachebo Ring transactions from the Cachebo destined for the AK ring. This is commonly used for credit returns and GO responses.
BL_CACHE	bxxxxx1xx	BL - Cacheno Ring transactions from the Cachebo destined for the BL ring. This is commonly used to send data from the cache to various destinations.
IV_CACHE	bxxxx1xxx	IV - Cachebo Ring transactions from the Cachebo destined for the IV ring. This is commonly used for snoops to the cores.
AD_CORE	bxxx1xxxx	AD - Corebo Ring transactions from the Corebo destined for the AD ring. This is commonly used for outbound requests.

Table 2-41. Unit Masks for TxR_INSERTS

Extension	umask [15:8]	Description
AK_CORE	bxx1xxxxx	AK - Corebo Ring transactions from the Corebo destined for the AK ring. This is commonly used for snoop responses coming from the core and destined for a Cachebo.
BL_CORE	bx1xxxxxx	BL - Corebo Ring transactions from the Corebo destined for the BL ring. This is commonly used for transfering writeback data to the cache.

2.4 Home Agent (HA) Performance Monitoring

Each HA is responsible for the protocol side of memory interactions, including coherent and non-coherent home agent protocols (as defined in the *Intel® QuickPath Interconnect Specification*). Additionally, the HA is responsible for ordering memory reads/writes, coming in from the modular Ring, to a given address such that the IMC (memory controller).

In other words, it is the coherency agent responsible for guarding the memory controller. All requests for memory attached to the coupled IMC must first be ordered through the HA. As such, it provides several functions:

- Interface between Ring and IMC:
  Regardless of the memory technology, the Home Agent receives memory read and write requests from the modular ring. It checks the memory transaction type, detects and resolves the coherent conflict, and finally schedules a corresponding transaction to the memory controller. It is also responsible for returning the response and completion to the requester.
- Conflict Manager:
  All requests must go through conflict management logic in order to ensure coherent consistency. In other words, the view of data must be the same across all coherency agents regardless of who is reading or modifying the data. On Intel® QPI, the home agent is responsible for tracking all requests to a given address and ensuring that the results are consistent.
- Memory Access Ordering Control:
  The Home Agent guarantees the ordering of RAW, WAW and WAR.
- Home Snoop Protocol Support (for parts with Directory Support):
  The Home Agent supports Intel® QPI's home snoop protocol by initiating snoops on behalf of requests. Closely tied to the directory feature, the home agent has the ability to issue snoops to the peer caching agents for requests based on the directory information.
- Directory Support:
  In order to satisfy performance requirements for the 4 socket and scalable DP segments, the Intel® Xeon® Processor D-1500 Product Family Home Agent implements a snoop directory which tracks all cachelines residing behind this Home Agent. This directory is used to reduce the snoop traffic when Intel® QPI bandwidth would otherwise be strained. The directory is not intended for typical 2S topologies.

2.4.1 HA Performance Monitoring Overview

Each HA Box supports event monitoring through four 48-bit wide counters (HAn_PCI_PMON_CTR{3:0}). Each of these counters can be programmed (HAn_PCI_PMON_CTL{3:0}) to capture any HA event. The HA counters will increment by a maximum of 128b per cycle.

For information on how to setup a monitoring session, refer to Section 2.1, "Uncore Per-Socket Performance Monitoring Control".

2.4.1.1 HA PMON Registers - On Overflow and the Consequences (PMI/Freeze)

If an overflow is detected from an HA performance counter, the overflow bit is set at the box level (HAn_PCI_PMON_BOX_STATUS.ov). If the counter is enabled to communicate the overflow (HAn_PCI_PMON_CTL.ov_en is set to 1), an overflow message is sent to the UBox. When the UBox receives the overflow signal, the U_MSR_PMON_GLOBAL_STATUS.ov_h bit corresponding to the HA generating the overflow is set (see Table 2-3, "U_MSR_PMON_GLOBAL_STATUS Register – Field Definitions"), a global freeze signal is sent and a PMI can be generated.

Once a freeze has occurred, in order to see a new freeze, the overflow responsible for the freeze must be cleared by setting the corresponding bit in HAn_PCI_PMON_BOX_STATUS.ov and U_MSR_PMON_GLOBAL_STATUs.ov_hn to 1 (which acts to clear the bits). Assuming all the counters have been locally enabled (*.en* bit in control registers meant to monitor events) and the overflow bits have been cleared, the HA is prepared for a new sample interval. Once the global controls have been re-enabled (Section 2.1.4, "Enabling a New Sample Interval from Frozen Counters"), counting will resume.

Table 2-42. HA Performance Monitoring Registers (PCICFG)

Register Name	PCICFG Address	Size (bits)	Description
PCICFG Base Address	Dev:Func DeviceID		
HA0 PMON Registers	D18:F1 0x6F30		
HA1 PMON Registers	D18:F5 0x6F38		
Box-Level Control/Status			
HAn_PCI_PMON_BOX_STATUS	F8	32	HA n PMON Box-Wide Status
HAn_PCI_PMON_BOX_CTL	F4	32	HA n PMON Box-Wide Control
Generic Counter Control			
HAn_PCI_PMON_CTL3	E4	32	HA n PMON Control for Counter 3
HAn_PCI_PMON_CTL2	E0	32	HA n PMON Control for Counter 2
HAn_PCI_PMON_CTL1	DC	32	HA n PMON Control for Counter 1
HAn_PCI_PMON_CTL0	D8	32	HA n PMON Control for Counter 0
Generic Counters			
HAn_PCI_PMON_CTR3	BC+B8	32x2	HA n PMON Counter 3
HAn_PCI_PMON_CTR2	B4+B0	32x2	HA n PMON Counter 2
HAn_PCI_PMON_CTR1	AC+A8	32x2	HA n PMON Counter 1
HAn_PCI_PMON_CTR0	A4+A0	32x2	HA n PMON Counter 0
Box-Level Filter			
HAn_PCI_PMON_BOX_OPCODEMATCH	48	32	HA n PMON Opcode Match
HAn_PCI_PMON_BOX_ADDRMATCH1	44	32	HA n PMON Address Match 1
HAn_PCI_PMON_BOX_ADDRMATCH0	40	32	HA n PMON Address Match 0

2.4.1.2 HA Box Level PMON State

The following registers represent the state governing all box-level PMUs in the HA Box.

In the case of the HA, the HAn_PCI_PMON_BOX_CTL register provides the ability to manually freeze the counters in the box (*.frz*) and reset the generic state (*.rst_ctrs* and *.rst_ctrl*).

If an overflow is detected from one of the HA PMON registers, the corresponding bit in the HAn_PCI_PMON_BOX_STATUS.ov field will be set. To reset these overflow bits, a user must write a value of '1' to them (which will clear the bits).

Table 2-43. HAn_PCI_PMON_BOX_CTL Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
ig	31:18	RV	0	Ignored
rsv	17:16	RV	0	Reserved; SW must write to 1 else behavior is undefined.
ig	15:9	RV	0	Ignored
frz	8	WO	0	Freeze. If set to 1 the counters in this box will be frozen.
ig	7:2	RV	0	Ignored
rst_ctrs	1	WO	0	Reset Counters. When set to 1, the Counter Registers will be reset to 0.
rst_ctrl	0	WO	0	Reset Control. When set to 1, the Counter Control Registers will be reset to 0.

Table 2-44. HAn_PCI_PMON_BOX_STATUS Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
ig	31:5	RV	0	Ignored
rsv	4	RV	0	Reserved; SW must write to 0 else behavior is undefined.
ov	3:0	RW1C	0	If an overflow is detected from the corresponding HAn_PCI_PMON_CTR register, it's overflow bit will be set. NOTE: Write of '1' will clear the bit.

2.4.1.3 HA PMON state - Counter/Control Pairs

The following table defines the layout of the HA performance monitor control registers. The main task of these configuration registers is to select the event to be monitored by their respective data counter (*.ev_sel*, *.umask*). Additional control bits are provided to shape the incoming events (e.g. *.invert*, *.edge_det*, *.thresh*) as well as provide additional functionality for monitoring software (*.rst,.ov_en*).

Table 2-45. HAn_PCI_PMON_CTL{3-0} Register – Field Definitions (Sheet 1 of 2)

Field	Bits	Attr	HW Reset Val	Description
thresh	31:24	RW-V	0	Threshold used in counter comparison.
invert	23	RW-V	0	Invert comparison against Threshold. 0 - comparison will be 'is event increment >= threshold?'. 1 - comparison is inverted - 'is event increment < threshold?' NOTE: .invert is in series following .thresh, Due to this, the .thresh field must be set to a non-0 value. For events that increment by no more than 1 per cycle, set .thresh to 0x1. Also, if .edge_det is set to 1, the counter will increment when a 1 to 0 transition (i.e. falling edge) is detected.
en	22	RW-V	0	Local Counter Enable.

Table 2-45. HAn_PCI_PMON_CTL{3-0} Register – Field Definitions (Sheet 2 of 2)

Field	Bits	Attr	HW Reset Val	Description
rsv	21	RV	0	Reserved. SW must write to 0 else behavior is undefined.
ov_en	20	RW-V	0	When this bit is asserted and the corresponding counter overflows, its overflow bit is set in the local status register (HA_PCI_PMON_BOX_STATUS.ov) and an overflow is sent on the message channel to the UBox. When the overflow is received by the UBox, the bit corresponding to this HA will be set in U_MSR_PMON_GLOBAL_STATUS.ov_h{1,0}.
ig	19	RV	0	Ignored
edge_det	18	RW-V	0	When set to 1, rather than measuring the event in each cycle it is active, the corresponding counter will increment when a 0 to 1 transition (i.e. rising edge) is detected. When 0, the counter will increment in each cycle that the event is asserted. NOTE: .edge_det is in series following .thresh, Due to this, the .thresh field must be set to a non-0 value. For events that increment by no more than 1 per cycle, set .thresh to 0x1.
rst	17	WO	0	When set to 1, the corresponding counter will be cleared to 0.
rsv	16	RV	0	Reserved. SW must write to 0 else behavior is undefined.
umask	15:8	RW-V	0	Select subevents to be counted within the selected event.
ev_sel	7:0	RW-V	0	Select event to be counted.

The HA performance monitor data registers are 48-bit wide. A counter overflow occurs when a carry out from bit 47 is detected. Software can force all uncore counting to freeze after N events by preloading a monitor with a count value of 2^{48} - N and setting the control register to send an overflow message to the UBox (refer to Section 2.1.1, "Counter Overflow"). During the interval of time between overflow and global disable, the counter value will wrap and continue to collect events.

If accessible, software can continuously read the data registers without disabling event collection.

Table 2-46. HA_PCI_PMON_CTR{3-0} Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
ig	63:48	RV	0	Ignored
event_count	47:0	RW-V	0	48-bit performance event counter

In addition to generic event counting, each HA provides a pair of Address Match registers and an Opcode Match register that allow a user to filter incoming packet traffic according to the packet Opcode, Message Class and Physical Address. The ADDR_OPC_MATCH.FILT event is provided to capture the filter match as an event. The fields are laid out as follows:

Table 2-47. HA_PCI_PMON_BOX_OPCODEMATCH Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
ig	31:6	RV	0	Ignored
opc	5:0	RWS	0	Match to this incoming (? which polarity?) opcode [5:4] Message Class - Encoded version 00 - HOM0 01 - HOM1 10 - NDR 11 - SNP [3:0] Intel QPI Opcode

Table 2-48. HA_PCI_PMON_BOX_ADDRMATCH1 Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
ig	31:14	RV	0	Ignored
hi_addr	13:0	RWS	0	Match to this System Address - Most Significant 14b of cache aligned address [45:32]

Table 2-49. HA_PCI_PMON_BOX_ADDRMATCH0 Register – Field Definitions

Field	Bits	HW Reset Val	HW Reset Val	Description
lo_addr	31:6	RWS	0	Match to this System Address - Least Significant 26b of cache aligned address [31:6]
ig	5:0	RV	0	Ignored

Note: The address comparison always ignores the lower 12 bits of the physical address, even if they system is interleaving between sockets at the cache-line level. Therefore, this mask will always match to an OS virtual page, even if only a fraction of that page is mapped to the Home Agent under investigation. The mask is not adjusted for large pages, so matches will only be allowed within 4K granularity.

2.4.2 HA Performance Monitoring Events

The performance monitoring events within the HA include all events internal to the HA as well as events which track ring related activity at the HA ring stops. Internal events include the ability to track Directory Activity, Direct2Core Activity, IMC Read/Write Traffic, time spent dealing with Conflicts, etc.

Other notable event types:

- IMC RPQ/WPQ Events
  Determine cycles the HA is stuck without credits in to the iMCs read/write queues.
- Ring Stop Events
  To track Egress and ring utilization (broken down by direction and ring type) statistics, as well as ring credits between the HA and Intel® QPI.
- Local/Remote Filtering

It was not possible to filter many of the events in NHM based on whether they originated from a local or remote caching agent. Many of the Intel® Xeon® Processor D-1500 Product Family events will be extended to support this.

- Snoop Latency

2.4.2.1 On the Major HA Structures:

The 128-entry **TF** (Tracker File) holds all transactions that arrive in the HA from the time they arrive until they are completed and leave the HA. Transactions could stay in this structure much longer than they are needed. TF is the critical resource each transaction needs before being sent to the IMC (memory controller)

TF average occupancy == (valid cnt * 128 / cycles)

TF average latency == (valid cnt * 128 / inserts)

Other Internal HA Queues of Interest:

TxR (aka EGR) - The HA has Egress (responses) queues for each ring (AD, AK, BL) as well as queues to track credits the HA has to push traffic onto those rings.

2.4.3 HA Box Events Ordered By Code

The following table summarizes the directly measured HA Box events.

Symbol Name	Event Code	Ctrs	Max Inc/Cyc	Description
CLOCKTICKS	0x00	0-3	1	uclks
REQUESTS	0x01	0-3	1	Read and Write Requests
TRACKER_CYCLES_FULL	0x02	0-3	1	Tracker Cycles Full
TRACKER_CYCLES_NE	0x03	0-3	1	Tracker Cycles Not Empty
TRACKER_OCCUPANCY	0x04	0-3	128	Tracker Occupancy Accumultor
TRACKER_PENDING_OCCUPANCY	0x05	0-3	127	Data Pending Occupancy Accumultor
SNOOP_CYCLES_NE	0x08	0-3	1	Cycles with Snoops Outstanding
SNOOP_OCCUPANCY	0x09	0-3	127	Tracker Snoops Outstanding Accumulator
SNOOPS_RSP_AFTER_DATA	0x0a	0-3	127	Data beat the Snoop Responses
CONFLICT_CYCLES	0x0b	1	0	Conflict Checks
DIRECTORY_LOOKUP	0x0c	0-3	1	Directory Lookups
DIRECTORY_UPDATE	0x0d	0-3	1	Directory Updates
TxR_AK	0x0e	0-3	1	Outbound Ring Transactions on AK
TxR_BL	0x10	0-3	1	Outbound DRS Ring Transactions to Cache
DIRECT2CORE_COUNT	0x11	0-3	1	Direct2Core Messages Sent
DIRECT2CORE_CYCLES_DISABLED	0x12	0-3	1	Cycles when Direct2Core was Disabled
DIRECT2CORE_TXN_OVERRIDE	0x13	0-3	1	Number of Reads that had Direct2Core Overridden
BYPASS_IMC	0x14	0-3	1	HA to iMC Bypass
RPQ_CYCLES_NO_REG_CREDITS	0x15	0-3	4	iMC RPQ Credits Empty - Regular
IMC_READS	0x17	0-3	4	HA to iMC Normal Priority Reads Issued
WPQ_CYCLES_NO_REG_CREDITS	0x18	0-3	4	HA iMC CHN0 WPQ Credits Empty - Regular
IMC_WRITES	0x1a	0-3	1	HA to iMC Full Line Writes Issued

Symbol Name	Event Code	Ctrs	Max Inc/Cyc	Description
TAD_REQUESTS_G0	0x1b	0-3	2	HA Requests to a TAD Region - Group 0
TAD_REQUESTS_G1	0x1c	0-3	2	HA Requests to a TAD Region - Group 1
IMC_RETRY	0x1e	0-3	1	Retry Events
ADDR_OPC_MATCH	0x20	0-3	1	Intel QPI Address/Opcode Match
SNOOP_RESP	0x21	0-3	1	Snoop Responses Received
IGR_NO_CREDIT_CYCLES	0x22	0-3	1	Cycles without Intel QPI Ingress Credits
TxR_AD_CYCLES_FULL	0x2a	0-3	1	AD Egress Full
TxR_AK_CYCLES_FULL	0x32	0-3	1	AK Egress Full
TxR_BL_OCCUPANCY	0x34	0-3	20	BL Egress Occupancy
TxR_BL_CYCLES_FULL	0x36	0-3	1	BL Egress Full
RING_AD_USED	0x3e	0-3	1	HA AD Ring in Use
RING_AK_USED	0x3f	0-3	1	HA AK Ring in Use
RING_BL_USED	0x40	0-3	1	HA BL Ring in Use
DIRECTORY_LAT_OPT	0x41	0-3	1	Directory Lat Opt Return
BT_CYCLES_NE	0x42	0-3	1	BT Cycles Not Empty
BT_OCCUPANCY	0x43	0-3	512	BT Occupancy
OSB	0x53	0-3	1	OSB Snoop Broadcast
OSB_EDR	0x54	0-3	1	OSB Early Data Return
SNP_RESP_RECV_LOCAL	0x60	0-3	1	Snoop Responses Received Local
TxR_STARVED	0x6d	0-3	1	Injection Starvation
HITME_LOOKUP	0x70	0-3	1	Counts Number of times HitMe Cache is accessed
HITME_HIT	0x71	0-3	1	Counts Number of Hits in HitMe Cache
HITME_HIT_PV_BITS_SET	0x72	0-3	1	Accumulates Number of PV bits set on HitMe Cache Hits

2.4.4 HA Box Common Metrics (Derived Events)

The following table summarizes metrics commonly calculated from HA Box events.

Symbol Name: Definition	Equation
HITME_INSERTS:	HITME_LOOKUP.ALLOCS - HITME_HITS.ALLOCS
HITME_INVAL:	HITME_HIT.INVALS
PCT_CYCLES_BL_FULL: Percentage of time the BL Egress Queue is full	TxR_BL_CYCLES_FULL.ALL / SAMPLE_INTERVAL
PCT_CYCLES_D2C_DISABLED: Percentage of time that Direct2Core was disabled.	DIRECT2CORE_CYCLES_DISABLED / SAMPLE_INTERVAL
PCT_RD_REQUESTS: Percentage of HA traffic that is from Read Requests	REQUESTS.READS / (REQUESTS.READS + REQUESTS.WRITES)
PCT_WR_REQUESTS: Percentage of HA traffic that is from Write Requests	REQUESTS.WRITES / (REQUESTS.READS + REQUESTS.WRITES)

2.4.5 HA Box Performance Monitor Event List

The section enumerates Intel® Xeon® Processor D-1500 Product Family performance monitoring events for the HA Box.

ADDR_OPC_MATCH

- **Title:** Intel QPI Address/Opcode Match
- **Category:** ADDR_OPCODE_MATCH Events
- **Event Code:** 0x20
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-50. Unit Masks for ADDR_OPC_MATCH

Extension	umask [15:8]	Filter Dep	Description
ADDR	bxxxxxxx1	HA_AddrMatch0[31:6], HA_AddrMatch1[13:0]	Address
OPC	bxxxxxx1x	HA_OpcodeMatch[5:0]	Opcode
FILT	b00000011	HA_AddrMatch0[31:6], HA_AddrMatch1[13:0], HA_OpcodeMatch[5:0]	Address & Opcode Match
AD	bxxxxx1xx	HA_OpcodeMatch[5:0]	AD Opcodes
BL	bxxxx1xxx	HA_OpcodeMatch[5:0]	BL Opcodes
AK	bxxx1xxxx	HA_OpcodeMatch[5:0]	AK Opcodes

BT_CYCLES_NE

- **Title:** BT Cycles Not Empty
- **Category:** BT (Backup Tracker) Events
- **Event Code:** 0x42
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Cycles the Backup Tracker (BT) is not empty.
- **NOTE:** Will not count case HT is empty and a Bypass happens.

BT_OCCUPANCY

- **Title:** BT Occupancy
- **Category:** BT (Backup Tracker) Events
- **Event Code:** 0x43
- **Max. Inc/Cyc:.** 512, **Register Restrictions:** 0-3
- **Definition:** Accumulates the occupancy of the HA BT pool in every cycle. This can be used with the "not empty" stat to calculate average queue occupancy or the "allocations" stat in order to calculate average queue latency. HA BTs are allocated as soon as a request enters the HA and is released after the snoop response and data return (or post in the case of a write) and the response is returned on the ring.

BYPASS_IMC

- **Title:** HA to iMC Bypass
- **Category:** BYPASS Events
- **Event Code:** 0x14
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when the HA was able to bypass was attempted. This is a latency optimization for situations when there is light loadings on the memory subsystem. This can be filted by when the bypass was taken and when it was not.
- **NOTE:** Only read transactions use iMC bypass

Table 2-52. Unit Masks for BYPASS_IMC

Extension	umask [15:8]	Description
TAKEN	bxxxxxxx1	Taken Filter for transactions that succeeded in taking the bypass.
NOT_TAKEN	bxxxxxx1x	Not Taken Filter for transactions that could not take the bypass.

CLOCKTICKS

- **Title:** uclks
- **Category:** UCLK Events
- **Event Code:** 0x00
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of uclks in the HA. This will be slightly different than the count in the Ubox because of enable/freeze delays. The HA is on the other side of the die from the fixed Ubox uclk counter, so the drift could be somewhat larger than in units that are closer like the Intel QPI Agent.

CONFLICT_CYCLES

- **Title:** Conflict Checks
- **Category:** CONFLICTS Events
- **Event Code:** 0x0b
- **Max. Inc/Cyc:.** 0, **Register Restrictions:** 1
- **Filter Dependency:** N
- **Definition:**

DIRECT2CORE_COUNT

- **Title:** Direct2Core Messages Sent
- **Category:** DIRECT2CORE Events
- **Event Code:** 0x11
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of Direct2Core messages sent
- **NOTE:** Will not be implemented since OUTBOUND_TX_BL:0x1 will count DRS to CORE which is effectively the same thing as D2C count

DIRECT2CORE_CYCLES_DISABLED

- **Title:** Cycles when Direct2Core was Disabled
- **Category:** DIRECT2CORE Events
- **Event Code:** 0x12
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3

- **Definition:** Number of cycles in which Direct2Core was disabled

DIRECT2CORE_TXN_OVERRIDE

- **Title:** Number of Reads that had Direct2Core Overridden
- **Category:** DIRECT2CORE Events
- **Event Code:** 0x13
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of Reads where Direct2Core overridden

DIRECTORY_LAT_OPT

- **Title:** Directory Lat Opt Return
- **Category:** DIRECTORY Events
- **Event Code:** 0x41
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Directory Latency Optimization Data Return Path Taken. When directory mode is enabled and the directory retuned for a read is Dir=I, then data can be returned using a faster path if certain conditions are met (credits, free pipeline, etc).

DIRECTORY_LOOKUP

- **Title:** Directory Lookups
- **Category:** DIRECTORY Events
- **Event Code:** 0x0c
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of transactions that looked up the directory. Can be filtered by requests that had to snoop and those that did not have to.
- **NOTE:** Only valid for parts that implement the Directory

Table 2-54. Unit Masks for DIRECTORY_LOOKUP

Extension	umask [15:8]	Description
SNP	bxxxxxxx1	Snoop Needed Filters for transactions that had to send one or more snoops because the directory bit was set.
NO_SNP	bxxxxxx1x	Snoop Not Needed Filters for transactions that did not have to send any snoops because the directory bit was clear.

DIRECTORY_UPDATE

- **Title:** Directory Updates
- **Category:** DIRECTORY Events
- **Event Code:** 0x0d
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of directory updates that were required. These result in writes to the memory controller. This can be filtered by directory sets and directory clears.
- **NOTE:** Only valid for parts that implement the Directory

Table 2-55. Unit Masks for DIRECTORY_UPDATE

Extension	umask [15:8]	Description
SET	bxxxxxxx1	Directory Set Filter for directory sets. This occurs when a remote read transaction requests memory, bringing it to a remote cache.
CLEAR	bxxxxxx1x	Directory Clear Filter for directory clears. This occurs when snoops were sent and all returned with RspI.
ANY	bxxxxxx11	Any Directory Update

HITME_HIT

- **Title:** Counts Number of Hits in HitMe Cache
- **Category:** HitME Events
- **Event Code:** 0x71
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-56. Unit Masks for HITME_HIT

Extension	umask [15:8]	Description
READ_OR_INVITOE	bxxxxxxx1	op is RdCode, RdData, RdDataMigratory, RdInvOwn, RdCur or InvItoE
WBMTOI	bxxxxxx1x	op is WbMtoI
ACKCNFLTWBI	bxxxxx1xx	op is AckCnfltWbI
WBMTOE_OR_S	bxxxx1xxx	op is WbMtoE or WbMtoS
HOM	b00001111	HOM Requests
RSPFWDI_REMOTE	bxxx1xxxx	op is RspIFwd or RspIFwdWb for a remote request
RSPFWDI_LOCAL	bxx1xxxxx	op is RspIFwd or RspIFwdWb for a local request
INVALS	b00100110	Invalidations
RSPFWDS	bx1xxxxxx	op is RsSFwd or RspSFwdWb
EVICTS	b01000010	Allocations
ALLOCS	b01110000	Allocations
RSP	b1xxxxxxx	op is RspI, RspIWb, RspS, RspSWb, RspCnflt or RspCnfltWbI
ALL	b11111111	All Requests

HITME_HIT_PV_BITS_SET

- **Title:** Accumulates Number of PV bits set on HitMe Cache Hits
- **Category:** HitME Events
- **Event Code:** 0x72
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-57. Unit Masks for HITME_HIT_PV_BITS_SET

Extension	umask [15:8]	Description
READ_OR_INVITOE	bxxxxxxx1	op is RdCode, RdData, RdDataMigratory, RdInvOwn, RdCur or InvItoE
WBMTOI	bxxxxxx1x	op is WbMtoI

Table 2-57. Unit Masks for HITME_HIT_PV_BITS_SET

Extension	umask [15:8]	Description
ACKCNFLTWBI	bxxxxx1xx	op is AckCnfltWbI
WBMTOE_OR_S	bxxxx1xxx	op is WbMtoE or WbMtoS
HOM	b00001111	HOM Requests
RSPFWDI_REMOTE	bxxx1xxxx	op is RspIFwd or RspIFwdWb for a remote request
RSPFWDI_LOCAL	bxx1xxxxx	op is RspIFwd or RspIFwdWb for a local request
RSPFWDS	bx1xxxxxx	op is RsSFwd or RspSFwdWb
RSP	b1xxxxxxx	op is RspI, RspIWb, RspS, RspSWb, RspCnflt or RspCnfltWbI
ALL	b11111111	All Requests

HITME_LOOKUP

- **Title:** Counts Number of times HitMe Cache is accessed
- **Category:** HitME Events
- **Event Code:** 0x70
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-58. Unit Masks for HITME_LOOKUP

Extension	umask [15:8]	Description
READ_OR_INVITOE	bxxxxxxx1	op is RdCode, RdData, RdDataMigratory, RdInvOwn, RdCur or InvItoE
WBMTOI	bxxxxxx1x	op is WbMtoI
ACKCNFLTWBI	bxxxxx1xx	op is AckCnfltWbI
WBMTOE_OR_S	bxxxx1xxx	op is WbMtoE or WbMtoS
HOM	b00001111	HOM Requests
RSPFWDI_REMOTE	bxxx1xxxx	op is RspIFwd or RspIFwdWb for a remote request
RSPFWDI_LOCAL	bxx1xxxxx	op is RspIFwd or RspIFwdWb for a local request
INVALS	b00100110	Invalidations
RSPFWDS	bx1xxxxxx	op is RsSFwd or RspSFwdWb
ALLOCS	b01110000	Allocations
RSP	b1xxxxxxx	op is RspI, RspIWb, RspS, RspSWb, RspCnflt or RspCnfltWbI
ALL	b11111111	All Requests

IGR_NO_CREDIT_CYCLES

- **Title:** Cycles without Intel QPI Ingress Credits
- **Category:** QPI_IGR_CREDITS Events
- **Event Code:** 0x22
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles when the HA does not have credits to send messages to the Intel QPI Agent. This can be filtered by the different credit pools and the different links.

Table 2-59. Unit Masks for IGR_NO_CREDIT_CYCLES

Extension	umask [15:8]	Description
AD_QPI0	bxxxxxxx1	AD to Intel QPI Link 0
AD_QPI1	bxxxxxx1x	AD to Intel QPI Link 1
BL_QPI0	bxxxxx1xx	BL to Intel QPI Link 0
BL_QPI1	bxxxx1xxx	BL to Intel QPI Link 1
AD_QPI2	bxxx1xxxx	BL to Intel QPI Link 0
BL_QPI2	bxx1xxxxx	BL to Intel QPI Link 1

IMC_READS

- **Title:** HA to iMC Normal Priority Reads Issued
- **Category:** IMC_READS Events
- **Event Code:** 0x17
- **Max. Inc/Cyc:.** 4, **Register Restrictions:** 0-3
- **Definition:** Count of the number of reads issued to any of the memory controller channels. This can be filtered by the priority of the reads.
- **NOTE:** Does not count reads using the bypass path. That is counted separately in HA_IMC.BYPASS

Table 2-60. Unit Masks for IMC_READS

Extension	umask [15:8]	Description
NORMAL	b00000001	Normal Priority

IMC_RETRY

- **Title:** Retry Events
- **Category:** IMC_MISC Events
- **Event Code:** 0x1e
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

IMC_WRITES

- **Title:** HA to iMC Full Line Writes Issued
- **Category:** IMC_WRITES Events
- **Event Code:** 0x1a
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the total number of full line writes issued from the HA into the memory controller. This counts for all four channels. It can be filtered by full/partial and ISOCH/non-ISOCH.

Table 2-61. Unit Masks for IMC_WRITES

Extension	umask [15:8]	Description
FULL	bxxxxxxx1	Full Line Non-ISOCH
PARTIAL	bxxxxxx1x	Partial Non-ISOCH
FULL_ISOCH	bxxxxx1xx	ISOCH Full Line
PARTIAL_ISOCH	bxxxx1xxx	ISOCH Partial
ALL	b00001111	All Writes

OSB

- **Title:** OSB Snoop Broadcast
- **Category:** OSB (Opportunistic Snoop Broadcast) Events
- **Event Code:** 0x53
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Count of OSB snoop broadcasts. Counts by 1 per request causing OSB snoops to be broadcast. Does not count all the snoops generated by OSB.

Table 2-62. Unit Masks for OSB

Extension	umask [15:8]	Description
READS_LOCAL	bxxxxxx1x	Local Reads
INVITOE_LOCAL	bxxxxx1xx	Local InvItoE
REMOTE	bxxxx1xxx	Remote
CANCELLED	bxxx1xxxx	Cancelled OSB Snoop broadcast cancelled due to D2C or Other. OSB cancel is counted when OSB local read is not allowed even when the transaction in local InItoE. It also counts D2C OSB cancel, but also includes the cases were D2C was not set in the first place for the transaction coming from the ring.
READS_LOCAL_USEFUL	bxx1xxxxx	Reads Local - Useful
REMOTE_USEFUL	bx1xxxxxx	Remote - Useful

OSB_EDR

- **Title:** OSB Early Data Return
- **Category:** OSB (Opportunistic Snoop Broadcast) Events
- **Event Code:** 0x54
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of transactions that broadcast snoop due to OSB, but found clean data in memory and was able to do early data return

Table 2-63. Unit Masks for OSB_EDR

Extension	umask [15:8]	Description
ALL	bxxxxxxx1	All
READS_LOCAL_I	bxxxxxx1x	Reads to Local I
READS_REMOTE_I	bxxxxx1xx	Reads to Remote I
READS_LOCAL_S	bxxxx1xxx	Reads to Local S
READS_REMOTE_S	bxxx1xxxx	Reads to Remote S

REQUESTS

- **Title:** Read and Write Requests
- **Category:** TRACKER Events
- **Event Code:** 0x01
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the total number of read requests made into the Home Agent. Reads include all read opcodes (including RFO). Writes include all writes (streaming, evictions, HitM, etc).

Table 2-64. Unit Masks for REQUESTS

Extension	umask [15:8]	Description
READS_LOCAL	bxxxxxxx1	Local Reads This filter includes only read requests coming from the local socket. This is a good proxy for LLC Read Misses (including RFOs) from the local socket.
READS_REMOTE	bxxxxxx1x	Remote Reads This filter includes only read requests coming from the remote socket. This is a good proxy for LLC Read Misses (including RFOs) from the remote socket.
READS	b00000011	Reads Incoming ead requests. This is a good proxy for LLC Read Misses (including RFOs).
WRITES_LOCAL	bxxxxx1xx	Local Writes This filter includes only writes coming from the local socket.
WRITES_REMOTE	bxxxx1xxx	Remote Writes This filter includes only writes coming from remote sockets.
WRITES	b00001100	Writes Incoming write requests.
INVITOE_LOCAL	bxxx1xxxx	Local InvItoEs This filter includes only InvItoEs coming from the local socket.
INVITOE_REMOTE	bxx1xxxxx	Remote InvItoEs This filter includes only InvItoEs coming from remote sockets.

RING_AD_USED

- **Title:** HA AD Ring in Use
- **Category:** RING Events
- **Event Code:** 0x3e
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles that the AD ring is being used at this ring stop. This includes when packets are passing by and when packets are being sunk, but does not include when packets are being sent from the ring stop.
- **NOTE:** In any cycle, a ring stop can see up to one packet moving in the CW direction and one packet moving in the CCW direction.

Table 2-65. Unit Masks for RING_AD_USED

Extension	umask [15:8]	Description
CW_EVEN	bxxxxxxx1	Clockwise and Even Filters for the Clockwise and Even ring polarity.
CW_ODD	bxxxxxx1x	Clockwise and Odd Filters for the Clockwise and Odd ring polarity.
CW	b00000011	Clockwise
CCW_EVEN	bxxxxx1xx	Counterclockwise and Even Filters for the Counterclockwise and Even ring polarity.
CCW_ODD	bxxxx1xxx	Counterclockwise and Odd Filters for the Counterclockwise and Odd ring polarity.
CCW	b00001100	Counterclockwise

RING_AK_USED

- **Title:** HA AK Ring in Use
- **Category:** RING Events
- **Event Code:** 0x3f
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles that the AK ring is being used at this ring stop. This includes when packets are passing by and when packets are being sunk, but does not include when packets are being sent from the ring stop.
- **NOTE:** In any cycle, a ring stop can see up to one packet moving in the CW direction and one packet moving in the CCW direction.

Table 2-66. Unit Masks for RING_AK_USED

Extension	umask [15:8]	Description
CW_EVEN	bxxxxxxx1	Clockwise and Even Filters for the Clockwise and Even ring polarity.
CW_ODD	bxxxxxx1x	Clockwise and Odd Filters for the Clockwise and Odd ring polarity.
CW	b00000011	Clockwise
CCW_EVEN	bxxxxx1xx	Counterclockwise and Even Filters for the Counterclockwise and Even ring polarity.
CCW_ODD	bxxxx1xxx	Counterclockwise and Odd Filters for the Counterclockwise and Odd ring polarity.
CCW	b00001100	Counterclockwise

RING_BL_USED

- **Title:** HA BL Ring in Use
- **Category:** RING Events
- **Event Code:** 0x40
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles that the BL ring is being used at this ring stop. This includes when packets are passing by and when packets are being sunk, but does not include when packets are being sent from the ring stop.
- **NOTE:** In any cycle, a ring stop can see up to one packet moving in the CW direction and one packet moving in the CCW direction.

Table 2-67. Unit Masks for RING_BL_USED

Extension	umask [15:8]	Description
CW_EVEN	bxxxxxxx1	Clockwise and Even Filters for the Clockwise and Even ring polarity.
CW_ODD	bxxxxxx1x	Clockwise and Odd Filters for the Clockwise and Odd ring polarity.
CW	b00000011	Clockwise
CCW_EVEN	bxxxxx1xx	Counterclockwise and Even Filters for the Counterclockwise and Even ring polarity.
CCW_ODD	bxxxx1xxx	Counterclockwise and Odd Filters for the Counterclockwise and Odd ring polarity.
CCW	b00001100	Counterclockwise

RPQ_CYCLES_NO_REG_CREDITS

- **Title:** iMC RPQ Credits Empty - Regular
- **Category:** RPQ_CREDITS Events
- **Event Code:** 0x15
- **Max. Inc/Cyc:.** 4, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles when there are no "regular" credits available for posting reads from the HA into the iMC. In order to send reads into the memory controller, the HA must first acquire a credit for the iMC's RPQ (read pending queue). This queue is broken into regular credits/buffers that are used by general reads, and "special" requests such as ISOCH reads. This count only tracks the regular credits Common high banwidth workloads should be able to make use of all of the regular buffers, but it will be difficult (and uncommon) to make use of both the regular and special buffers at the same time. One can filter based on the memory controller channel. One or more channels can be tracked at a given time.

Table 2-68. Unit Masks for RPQ_CYCLES_NO_REG_CREDITS

Extension	umask [15:8]	Description
CHN0	b00000001	Channel 0 Filter for memory controller channel 0 only.
CHN1	b00000010	Channel 1 Filter for memory controller channel 1 only.
CHN2	b00000100	Channel 2 Filter for memory controller channel 2 only.
CHN3	b00001000	Channel 3 Filter for memory controller channel 3 only.

SNOOPS_RSP_AFTER_DATA

- **Title:** Data beat the Snoop Responses
- **Category:** SNOOPS Events
- **Event Code:** 0x0a
- **Max. Inc/Cyc:.** 127, **Register Restrictions:** 0-3
- **Definition:** Counts the number of reads when the snoop was on the critical path to the data return.

Table 2-69. Unit Masks for SNOOPS_RSP_AFTER_DATA

Extension	umask [15:8]	Description
LOCAL	b00000001	Local Requests This filter includes only requests coming from the local socket.
REMOTE	b00000010	Remote Requests This filter includes only requests coming from remote sockets.

SNOOP_CYCLES_NE

- **Title:** Cycles with Snoops Outstanding
- **Category:** SNOOPS Events
- **Event Code:** 0x08
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts cycles when one or more snoops are outstanding.

Table 2-70. Unit Masks for SNOOP_CYCLES_NE

Extension	umask [15:8]	Description
LOCAL	bxxxxxxx1	Local Requests This filter includes only requests coming from the local socket.
REMOTE	bxxxxxx1x	Remote Requests This filter includes only requests coming from remote sockets.
ALL	b00000011	All Requests Tracked for snoops from both local and remote sockets.

SNOOP_OCCUPANCY

- **Title:** Tracker Snoops Outstanding Accumulator
- **Category:** SNOOPS Events
- **Event Code:** 0x09
- **Max. Inc/Cyc:.** 127, **Register Restrictions:** 0-3
- **Definition:** Accumulates the occupancy of either the local HA tracker pool that have snoops pending in every cycle. This can be used in conjection with the "not empty" stat to calculate average queue occupancy or the "allocations" stat in order to calculate average queue latency. HA trackers are allocated as soon as a request enters the HA if an HT (HomeTracker) entry is available and this occupancy is decremented when all the snoop responses have returned.

Table 2-71. Unit Masks for SNOOP_OCCUPANCY

Extension	umask [15:8]	Description
LOCAL	b00000001	Local Requests This filter includes only requests coming from the local socket.
REMOTE	b00000010	Remote Requests This filter includes only requests coming from remote sockets.

SNOOP_RESP

- **Title:** Snoop Responses Received
- **Category:** SNP_RESP Events
- **Event Code:** 0x21
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the total number of RspI snoop responses received. Whenever a snoops are issued, one or more snoop responses will be returned depending on the topology of the system. In systems larger than 2s, when multiple snoops are returned this will count all the snoops that are received. For example, if 3 snoops were issued and returned RspI, RspS, and RspSFwd; then each of these sub-events would increment by 1.

Table 2-72. Unit Masks for SNOOP_RESP

Extension	umask [15:8]	Description
RSPI	bxxxxxxx1	RspI Filters for snoops responses of RspI. RspI is returned when the remote cache does not have the data, or when the remote cache silently evicts data (such as when an RFO hits non-modified data).
RSPS	bxxxxxx1x	RspS Filters for snoop responses of RspS. RspS is returned when a remote cache has data but is not forwarding it. It is a way to let the requesting socket know that it cannot allocate the data in E state. No data is sent with S RspS.

Table 2-72. Unit Masks for SNOOP_RESP

Extension	umask [15:8]	Description
RSPIFWD	bxxxxx1xx	RspIFwd Filters for snoop responses of RspIFwd. This is returned when a remote caching agent forwards data and the requesting agent is able to acquire the data in E or M states. This is commonly returned with RFO transactions. It can be either a HitM or a HitFE.
RSPSFWD	bxxxx1xxx	RspSFwd Filters for a snoop response of RspSFwd. This is returned when a remote caching agent forwards data but holds on to its currentl copy. This is common for data and code reads that hit in a remote socket in E or F state.
RSP_WB	bxxx1xxxx	Rsp*WB Filters for a snoop response of RspIWB or RspSWB. This is returned when a non-RFO request hits in M state. Data and Code Reads can return either RspIWB or RspSWB depending on how the system has been configured. InvItoE transactions will also return RspIWB because they must acquire ownership.
RSP_FWD_WB	bxx1xxxxx	Rsp*Fwd*WB Filters for a snoop response of Rsp*Fwd*WB. This snoop response is only used in 4s systems. It is used when a snoop HITM's in a remote caching agent and it directly forwards data to a requestor, and simultaneously returns data to the home to be written back to memory.
RSPCNFLCT	bx1xxxxxx	RSPCNFLCT* Filters for snoops responses of RspConflict. This is returned when a snoop finds an existing outstanding transaction in a remote caching agent when it CAMs that caching agent. This triggers conflict resolution hardware. This covers both RspCnflct and RspCnflctWbI.

SNP_RESP_RECV_LOCAL

- **Title:** Snoop Responses Received Local
- **Category:** SNP_RESP Events
- **Event Code:** 0x60
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of snoop responses received for a Local request

Table 2-73. Unit Masks for SNP_RESP_RECV_LOCAL (Sheet 1 of 2)

Extension	umask [15:8]	Description
RSPI	bxxxxxxx1	RspI Filters for snoops responses of RspI. RspI is returned when the remote cache does not have the data, or when the remote cache silently evicts data (such as when an RFO hits non-modified data).
RSPS	bxxxxxx1x	RspS Filters for snoop responses of RspS. RspS is returned when a remote cache has data but is not forwarding it. It is a way to let the requesting socket know that it cannot allocate the data in E state. No data is sent with S RspS.
RSPIFWD	bxxxxx1xx	RspIFwd Filters for snoop responses of RspIFwd. This is returned when a remote caching agent forwards data and the requesting agent is able to acquire the data in E or M states. This is commonly returned with RFO transactions. It can be either a HitM or a HitFE.
RSPSFWD	bxxxx1xxx	RspSFwd Filters for a snoop response of RspSFwd. This is returned when a remote caching agent forwards data but holds on to its currentl copy. This is common for data and code reads that hit in a remote socket in E or F state.

Table 2-73. Unit Masks for SNP_RESP_RECV_LOCAL (Sheet 2 of 2)

Extension	umask [15:8]	Description
RSPxWB	bxxx1xxxx	Rsp*WB Filters for a snoop response of RspIWB or RspSWB. This is returned when a non-RFO request hits in M state. Data and Code Reads can return either RspIWB or RspSWB depending on how the system has been configured. InvItoE transactions will also return RspIWB because they must acquire ownership.
RSPxFWDxWB	bxx1xxxxx	Rsp*FWD*WB Filters for a snoop response of Rsp*Fwd*WB. This snoop response is only used in 4s systems. It is used when a snoop HITM's in a remote caching agent and it directly forwards data to a requestor, and simultaneously returns data to the home to be written back to memory.
RSPCNFLCT	bx1xxxxxx	RspCnflct Filters for snoops responses of RspConflict. This is returned when a snoop finds an existing outstanding transaction in a remote caching agent when it CAMs that caching agent. This triggers conflict resolution hardware. This covers both RspCnflct and RspCnflctWbI.
OTHER	b1xxxxxxx	Other Filters for all other snoop responses.

TAD_REQUESTS_G0

- **Title:** HA Requests to a TAD Region - Group 0
- **Category:** TAD Events
- **Event Code:** 0x1b
- **Max. Inc/Cyc:.** 2, **Register Restrictions:** 0-3
- **Definition:** Counts the number of HA requests to a given TAD region. There are up to 11 TAD (target address decode) regions in each home agent. All requests destined for the memory controller must first be decoded to determine which TAD region they are in. This event is filtered based on the TAD region ID, and covers regions 0 to 7. This event is useful for understanding how applications are using the memory that is spread across the different memory regions.

Table 2-74. Unit Masks for TAD_REQUESTS_G0

Extension	umask [15:8]	Description
REGION0	b00000001	TAD Region 0 Filters request made to TAD Region 0
REGION1	b00000010	TAD Region 1 Filters request made to TAD Region 1
REGION2	b00000100	TAD Region 2 Filters request made to TAD Region 2
REGION3	b00001000	TAD Region 3 Filters request made to TAD Region 3
REGION4	b00010000	TAD Region 4 Filters request made to TAD Region 4
REGION5	b00100000	TAD Region 5 Filters request made to TAD Region 5
REGION6	b01000000	TAD Region 6 Filters request made to TAD Region 6
REGION7	b10000000	TAD Region 7 Filters request made to TAD Region 7

TAD_REQUESTS_G1

- **Title:** HA Requests to a TAD Region - Group 1
- **Category:** TAD Events
- **Event Code:** 0x1c
- **Max. Inc/Cyc:.** 2, **Register Restrictions:** 0-3
- **Definition:** Counts the number of HA requests to a given TAD region. There are up to 11 TAD (target address decode) regions in each home agent. All requests destined for the memory controller must first be decoded to determine which TAD region they are in. This event is filtered based on the TAD region ID, and covers regions 8 to 10. This event is useful for understanding how applications are using the memory that is spread across the different memory regions.

Table 2-75. Unit Masks for TAD_REQUESTS_G1

Extension	umask [15:8]	Description
REGION8	b00000001	TAD Region 8 Filters request made to TAD Region 8
REGION9	b00000010	TAD Region 9 Filters request made to TAD Region 9
REGION10	b00000100	TAD Region 10 Filters request made to TAD Region 10
REGION11	b00001000	TAD Region 11 Filters request made to TAD Region 11

TRACKER_CYCLES_FULL

- **Title:** Tracker Cycles Full
- **Category:** TRACKER Events
- **Event Code:** 0x02
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles when the local HA tracker pool is completely used. This can be used with edge detect to identify the number of situations when the pool became fully utilized. This should not be confused with RTID credit usage -- which must be tracked inside each cbo individually -- but represents the actual tracker buffer structure. In other words, the system could be starved for RTIDs but not fill up the HA trackers. HA trackers are allocated as soon as a request enters the HA and is released after the snoop response and data return (or post in the case of a write) and the response is returned on the ring.

Table 2-76. Unit Masks for TRACKER_CYCLES_FULL

Extension	umask [15:8]	Description
GP	bxxxxxxx1	Cycles GP Completely Used Counts the number of cycles when the general purpose (GP) HA tracker pool (HT) is completely used. It will not return valid count when BT is disabled.
ALL	bxxxxxx1x	Cycles Completely Used Counts the number of cycles when the HA tracker pool (HT) is completely used including reserved HT entries. It will not return valid count when BT is disabled.

TRACKER_CYCLES_NE

- **Title:** Tracker Cycles Not Empty
- **Category:** TRACKER Events
- **Event Code:** 0x03
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles when the local HA tracker pool is not empty. This can be used with edge detect to identify the number of situations when the pool became empty. This

should not be confused with RTID credit usage -- which must be tracked inside each cbo individually -- but represents the actual tracker buffer structure. In other words, this buffer could be completely empty, but there may still be credits in use by the CBos. This stat can be used in conjunction with the occupancy accumulation stat in order to calculate average queue occpancy. HA trackers are allocated as soon as a request enters the HA if an HT (Home Tracker) entry is available and is released after the snoop response and data return (or post in the case of a write) and the response is returned on the ring.

Table 2-77. Unit Masks for TRACKER_CYCLES_NE

Extension	umask [15:8]	Description
LOCAL	bxxxxxxx1	Local Requests This filter includes only requests coming from the local socket.
REMOTE	bxxxxxx1x	Remote Requests This filter includes only requests coming from remote sockets.
ALL	b00000011	All Requests Requests coming from both local and remote sockets.

TRACKER_OCCUPANCY

- **Title:** Tracker Occupancy Accumultor
- **Category:** TRACKER Events
- **Event Code:** 0x04
- **Max. Inc/Cyc:.** 128, **Register Restrictions:** 0-3
- **Definition:** Accumulates the occupancy of the local HA tracker pool in every cycle. This can be used in conjection with the "not empty" stat to calculate average queue occupancy or the "allocations" stat in order to calculate average queue latency. HA trackers are allocated as soon as a request enters the HA if a HT (Home Tracker) entry is available and is released after the snoop response and data return (or post in the case of a write) and the response is returned on the ring.

Table 2-78. Unit Masks for TRACKER_OCCUPANCY

Extension	umask [15:8]	Description
READS_LOCAL	bxxxxx1xx	Local Read Requests
READS_REMOTE	bxxxx1xxx	Remote Read Requests
WRITES_LOCAL	bxxx1xxxx	Local Write Requests
WRITES_REMOTE	bxx1xxxxx	Remote Write Requests
INVITOE_LOCAL	bx1xxxxxx	Local InvItoE Requests
INVITOE_REMOTE	b1xxxxxxx	Remote InvItoE Requests

TRACKER_PENDING_OCCUPANCY

- **Title:** Data Pending Occupancy Accumultor
- **Category:** TRACKER Events
- **Event Code:** 0x05
- **Max. Inc/Cyc:.** 127, **Register Restrictions:** 0-3
- **Definition:** Accumulates the number of transactions that have data from the memory controller until they get scheduled to the Egress. This can be used to calculate the queuing latency for two things. (1) If the system is waiting for snoops, this will increase. (2) If the system can't schedule to the Egress because of either (a) Egress Credits or (b) Intel QPI BL IGR credits for remote requests.

Table 2-79. Unit Masks for TRACKER_PENDING_OCCUPANCY

Extension	umask [15:8]	Description
LOCAL	b00000001	Local Requests This filter includes only requests coming from the local socket.
REMOTE	b00000010	Remote Requests This filter includes only requests coming from remote sockets.

TxR_AD_CYCLES_FULL

- **Title:** AD Egress Full
- **Category:** EGRESS Events
- **Event Code:** 0x2a
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** AD Egress Full

Table 2-80. Unit Masks for TxR_AD_CYCLES_FULL

Extension	umask [15:8]	Description
SCHED0	bxxxxxxx1	Scheduler 0 Filter for cycles full from scheduler bank 0
SCHED1	bxxxxxx1x	Scheduler 1 Filter for cycles full from scheduler bank 1
ALL	bxxxxxx11	All Cycles full from both schedulers

TxR_AK

- **Title:** Outbound Ring Transactions on AK
- **Category:** OUTBOUND_TX Events
- **Event Code:** 0x0e
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

TxR_AK_CYCLES_FULL

- **Title:** AK Egress Full
- **Category:** EGRESS Events
- **Event Code:** 0x32
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** AK Egress Full

Table 2-82. Unit Masks for TxR_AK_CYCLES_FULL

Extension	umask [15:8]	Description
SCHED0	bxxxxxxx1	Scheduler 0 Filter for cycles full from scheduler bank 0
SCHED1	bxxxxxx1x	Scheduler 1 Filter for cycles full from scheduler bank 1
ALL	bxxxxxx11	All Cycles full from both schedulers

TxR_BL

- **Title:** Outbound DRS Ring Transactions to Cache
- **Category:** OUTBOUND_TX Events
- **Event Code:** 0x10
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of DRS messages sent out on the BL ring. This can be filtered by the destination.

Table 2-83. Unit Masks for TxR_BL

Extension	umask [15:8]	Description
DRS_CACHE	bxxxxxxx1	Data to Cache Filter for data being sent to the cache.
DRS_CORE	bxxxxxx1x	Data to Core Filter for data being sent directly to the requesting core.
DRS_QPI	bxxxxx1xx	Data to Intel QPI Filter for data being sent to a remote socket over Intel QPI.

TxR_BL_CYCLES_FULL

- **Title:** BL Egress Full
- **Category:** EGRESS Events
- **Event Code:** 0x36
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** BL Egress Full

Table 2-84. Unit Masks for TxR_BL_CYCLES_FULL

Extension	umask [15:8]	Description
SCHED0	bxxxxxxx1	Scheduler 0 Filter for cycles full from scheduler bank 0
SCHED1	bxxxxxx1x	Scheduler 1 Filter for cycles full from scheduler bank 1
ALL	bxxxxxx11	All Cycles full from both schedulers

TxR_BL_OCCUPANCY

- **Title:** BL Egress Occupancy
- **Category:** BL_EGRESS Events
- **Event Code:** 0x34
- **Max. Inc/Cyc:.** 20, **Register Restrictions:** 0-3
- **Definition:** BL Egress Occupancy

TxR_STARVED

- **Title:** Injection Starvation
- **Category:** EGRESS Events
- **Event Code:** 0x6d
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts injection starvation. This starvation is triggered when the Egress cannot send a transaction onto the ring for a long period of time.

Table 2-86. Unit Masks for TxR_STARVED

Extension	umask [15:8]	Description
AK	bxxxxxxx1	For AK Ring
BL	bxxxxxx1x	For BL Ring

WPQ_CYCLES_NO_REG_CREDITS

- **Title:** HA iMC CHN0 WPQ Credits Empty - Regular
- **Category:** WPQ_CREDITS Events
- **Event Code:** 0x18
- **Max. Inc/Cyc:.** 4, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles when there are no "regular" credits available for posting writes from the HA into the iMC. In order to send writes into the memory controller, the HA must first acquire a credit for the iMC's WPQ (write pending queue). This queue is broken into regular credits/buffers that are used by general writes, and "special" requests such as ISOCH writes. This count only tracks the regular credits Common high banwidth workloads should be able to make use of all of the regular buffers, but it will be difficult (and uncommon) to make use of both the regular and special buffers at the same time. One can filter based on the memory controller channel. One or more channels can be tracked at a given time.

Table 2-87. Unit Masks for WPQ_CYCLES_NO_REG_CREDITS

Extension	umask [15:8]	Description
CHN0	b00000001	Channel 0 Filter for memory controller channel 0 only.
CHN1	b00000010	Channel 1 Filter for memory controller channel 1 only.
CHN2	b00000100	Channel 2 Filter for memory controller channel 2 only.
CHN3	b00001000	Channel 3 Filter for memory controller channel 3 only.

2.5 Memory Controller (IMC) Performance Monitoring

The Intel® Xeon® Processor D-1500 Product Family integrated Memory Controller provides the interface to DRAM and communicates to the rest of the Uncore through the Home Agent (i.e. the IMC does not connect to the Ring).

In conjunction with the HA, the memory controller also provides a variety of RAS features.

2.5.1 Functional Overview

The memory controller is the interface between the home Home Agent (HA) and DRAM, translating read and write commands into specific memory commands and schedules them with respect to memory timing. The other main function of the memory controller is advanced ECC support.

Because of the data path affinity to the HA data path, the HA is paired with the memory controller.

The Intel® Xeon® Processor D-1500 Product Family supports up to 2 channels of DDR accessed through a single iMC. The number of DIMMs per channel depends on the speed it is running and the processor type.

A selection of IMC functionality that performance monitoring provides some insight into:

- Supports up to 16 ranks per channel with 8 independent banks per rank.
- ECC support (correct any error within a x4 device)
- Open or closed page policy
- ISOCH
- Demand and Patrol Scrubbing support
- Support for LR-DIMMs (load reduced) for a buffered memory solution demanding higher capacity memory subsytems.

2.5.2 IMC Performance Monitoring Overview

The IMC supports event monitoring through four 48-bit wide counters (MC_CHy_PCI_PMON_CTR{3:0}) and one fixed counter (MC_CHy_PCI_PMON_FIXED_CTR) for each DRAM channel (of which there are 2 in Intel® Xeon® Processor D-1500 Product Family) the MC is attached to. Each of these counters can be programmed (MC_CHy_PCI_PMON_CTL{3:0}) to capture any MC event. The MC counters will increment by a maximum of 8b per cycle.

For information on how to setup a monitoring session, refer to Section 2.1, “Uncore Per-Socket Performance Monitoring Control”.

2.5.2.1 IMC PMON Registers - On Overflow and the Consequences (PMI/Freeze)

If an overflow is detected from an MC performance counter, the overflow bit is set at the box level (MC_CHy_PCI_PMON_BOX_STATUS.ov). If the counter is enabled to communicate the overflow (MC_CHy_PCI_PMON_CTL.ov_en is set to 1), an overflow message is sent to the UBox. When the UBox receives the overflow signal, the U_MSR_PMON_GLOBAL_STATUS.ov_m bit corresponding to the MC generating the overflow is set (see Table 2-3, “U_MSR_PMON_GLOBAL_STATUS Register – Field Definitions”), a global freeze signal is sent and a PMI can be generated.

Once a freeze has occurred, in order to see a new freeze, the overflow responsible for the freeze must be cleared by setting the corresponding bit in MC_CHy_PCI_PMON_BOX_STATUS.ov and U_MSR_PMON_GLOBAL_STATUS.ov_mn to 1 (which acts to clear the bits). Assuming all the counters have been locally enabled (*.en* bit in control registers meant to monitor events) and the overflow bits have been cleared, the MC is prepared for a new sample interval. Once the global controls have been re-enabled (Section 2.1.4, "Enabling a New Sample Interval from Frozen Counters"), counting will resume.

2.5.3 IMC Performance Monitors

Table 2-88. IMC Performance Monitoring Registers (PCICFG) (Sheet 1 of 2)

Register Name	PCICFG Address	Size (bits)	Description
PCICFG Base Address	Dev:Func DeviceID		
MC0 Channel 0 PMON Registers	D20:F0 0x6FB4		
MC0 Channel 1 PMON Registers	D20:F1 0x6FB5		
MC0 Channel 2 PMON Registers	D21:F0 0x6FB0		
MC0 Channel 3 PMON Registers	D21:F1 0x6FB1		
MC1 Channel 0 PMON Registers	D23:F0 0x6FD4		
MC1 Channel 1 PMON Registers	D23:F1 0x6FD5		
MC1 Channel 2 PMON Registers	D24:F0 0x6FD0		
MC1 Channel 3 PMON Registers	D24:F1 0x6FD1		
Box-Level Control/Status			
MC_CHy_PCI_PMON_BOX_STATUS	F8	32	MC Channel y PMON Box-Wide Status
MC_CHy_PCI_PMON_BOX_CTL	F4	32	MC Channel y PMON Box-Wide Control
Generic Counter Control			
MC_CHy_PCI_PMON_FIXED_CTL	F0	32	MC Channel y PMON Control for Fixed Counter
MC_CHy_PCI_PMON_CTL3	E4	32	MC Channel y PMON Control for Counter 3
MC_CHy_PCI_PMON_CTL2	E0	32	MC Channel y PMON Control for Counter 2
MC_CHy_PCI_PMON_CTL1	DC	32	MC Channel y PMON Control for Counter 1
MC_CHy_PCI_PMON_CTL0	D8	32	MC Channel y PMON Control for Counter 0
Generic Counters			
MC_CHy_PCI_PMON_FIXED_CTR	D4+D0	32x2	MC Channel y PMON Fixed Counter

Table 2-88. IMC Performance Monitoring Registers (PCICFG) (Sheet 2 of 2)

Register Name	PCICFG Address	Size (bits)	Description
MC_CHy_PCI_PMON_CTR3	BC+B8	32x2	MC Channel y PMON Counter 3
MC_CHy_PCI_PMON_CTR2	B4+B0	32x2	MC Channel y PMON Counter 2
MC_CHy_PCI_PMON_CTR1	AC+A8	32x2	MC Channel y PMON Counter 1
MC_CHy_PCI_PMON_CTR0	A4+A0	32x2	MC Channel y PMON Counter 0

2.5.3.1 MC Box Level PMON State

The following registers represent the state governing all box-level PMUs in the MC Boxes.

In the case of the MC, the MC_CHy_PCI_PMON_BOX_CTL register provides the ability to manually freeze the counters in the box (*.frz*) and reset the generic state (*.rst_ctrs* and *.rst_ctrl*).

If an overflow is detected from one of the MC Box PMON registers, the corresponding bit in the MC_CHy_PCI_PMON_BOX_STATUS.ov field will be set. To reset these overflow bits, a user must write a value of '1' to them (which will clear the bits).

Table 2-89. MC_CHy_PCI_PMON_BOX_CTL Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
ig	31:18	RV	0	Ignored
rsv	17:16	RV	0	Reserved; SW must write to 1 else behavior is undefined.
ig	15:9	RV	0	Ignored
frz	8	WO	0	Freeze. If set to 1 the counters in this box will be frozen.
ig	7:2	RV	0	Ignored
rst_ctrs	1	WO	0	Reset Counters. When set to 1, the Counter Registers will be reset to 0.
rst_ctrl	0	WO	0	Reset Control. When set to 1, the Counter Control Registers will be reset to 0.

Table 2-90. MC_CHy_PCI_PMON_BOX_STATUS Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
ig	31:6	RV	0	Ignored
rsv	5	RV	0	Reserved. SW must write to 0 else behavior is undefined.
ov	4:0	RW1C	0	If an overflow is detected from the corresponding MC_CHy_PCI_PMON_CTR register, it's overflow bit will be set. NOTE: Write of '1' will clear the bit. Bit 4 -overflow for *_PMON_CTR4 Bit 1 -overflow for *_PMON_CTR1 Bit 0 -overflow for the fixed counter

2.5.3.2 MC PMON state - Counter/Control Pairs

The following table defines the layout of the MC performance monitor control registers. The main task of these configuration registers is to select the event to be monitored by their respective data counter (*.ev_sel*, *.umask*). Additional control bits are provided to shape the incoming events (e.g. *.invert*, *.edge_det*, *.thresh*) as well as provide additional functionality for monitoring software (*.rst*,*.ov_en*).

Table 2-91. MC_CHy_PCI_PMON_CTL{3-0} Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
thresh	31:24	RW-V	0	Threshold used in counter comparison.
invert	23	RW-V	0	Invert comparison against Threshold. 0 - comparison will be 'is event increment >= threshold?'. 1 - comparison is inverted - 'is event increment < threshold?' NOTE: .invert is in series following .thresh, Due to this, the .thresh field must be set to a non-0 value. For events that increment by no more than 1 per cycle, set .thresh to 0x1. Also, if .edge_det is set to 1, the counter will increment when a 1 to 0 transition (i.e. falling edge) is detected.
en	22	RW-V	0	Local Counter Enable.
rsv	21	RV	0	Reserved. SW must write to 0 else behavior is undefined.
ov_en	20	RW-V	0	When this bit is asserted and the corresponding counter overflows, its overflow bit is set in the local status register (MC_CHy_PCI_PMON_BOX_STATUS.ov) and an overflow is sent on the message channel to the UBox. When the overflow is received by the UBox, the bit corresponding to this MC will be set in U_MSR_PMON_GLOBAL_STATUS.ov_m{1,0}.
ig	19	RV	0	Ignored
edge_det	18	RW-V	0	When set to 1, rather than measuring the event in each cycle it is active, the corresponding counter will increment when a 0 to 1 transition (i.e. rising edge) is detected. When 0, the counter will increment in each cycle that the event is asserted. NOTE: .edge_det is in series following .thresh, Due to this, the .thresh field must be set to a non-0 value. For events that increment by no more than 1 per cycle, set .thresh to 0x1.
rst	17	WO	0	When set to 1, the corresponding counter will be cleared to 0.
rsv	16	RV	0	Reserved. SW must write to 0 else behavior is undefined.
umask	15:8	RW-V	0	Select subevents to be counted within the selected event.
ev_sel	7:0	RW-V	0	Select event to be counted.

All MC performance monitor data registers are 48-bit wide. A counter overflow occurs when a carry out from bit 47 is detected. Software can force all uncore counting to freeze after N events by preloading a monitor with a count value of 2^{48} - N and setting the control register to send an overflow message to the UBox (refer to Section 2.1.1, "Counter Overflow"). During the interval of time between overflow and global disable, the counter value will wrap and continue to collect events.

If accessible, software can continuously read the data registers without disabling event collection.

This is a counter that always tracks the number of DRAM clocks (dclks - half of DDR speed) in the IMC. The dclk never changes frequency (on a given system), and therefore is a good measure of wall clock (unlike the Uncore clock which can change frequency based on system load). This clock is generally a bit slower than the uclk (~800MHz to ~1.066GHz) and therefore has less fidelity.

Table 2-92. MC_CHy_PCI_PMON_FIXED_CTL Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
ig	31:24	RV	0	Ignored
invert	23	RW-V	0	Invert comparison against Threshold. 0 - comparison will be 'is event increment >= threshold?'. 1 - comparison is inverted - 'is event increment < threshold?' NOTE: .invert is in series following .thresh, Due to this, the .thresh field must be set to a non-0 value. For events that increment by no more than 1 per cycle, set .thresh to 0x1. Also, if .edge_det is set to 1, the counter will increment when a 1 to 0 transition (i.e. falling edge) is detected.
en	22	RW-V	0	Local Counter Enable.
rsv	21	RV	0	Reserved. SW must write to 0 else behavior is undefined.
ov_en	20	RW-V	0	When this bit is asserted and the corresponding counter overflows, a PMI exception is sent to the UBox.
rst	19	WO	0	When set to 1, the corresponding counter will be cleared to 0.
ig	18:0	RV	0	Ignored

Table 2-93. MC_CHy_PCI_PMON_CTR{FIXED,3-0} Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
ig	63:48	RV	0	Ignored
event_count	47:0	RW-V	0	48-bit performance event counter

2.5.4 IMC Performance Monitoring Events

A sampling of events available for monitoring in the IMC:

- **Translated commands**: Various Read and Write CAS commands
- **Memory commands**: CAS, Precharge, Refresh, Preemptions, etc,
- Page hits and page misses.
- **Page Closing** Events
- **Control of power consumption**: Thermal Throttling by Rank, Time spent in CKE ON mode, etc.

and many more.

Internal IMC Queues:

RPQ - Read Pending Queue. NOTE: HA also tracks some information related to the IMC's RPQ.

WPQ - Write Pending Queue. NOTE: HA also tracks some information related to the IMC's WPQ.

2.5.5 iMC Box Events Ordered By Code

The following table summarizes the directly measured iMC Box events.

Symbol Name	Event Code	Ctrs	Max Inc/Cyc	Description
DCLOCKTICKS	0x00	0-3	1	DRAM Clockticks
ACT_COUNT	0x01	0-3	1	DRAM Activate Count
PRE_COUNT	0x02	0-3	1	DRAM Precharge commands.
CAS_COUNT	0x04	0-3	1	DRAM RD_CAS and WR_CAS Commands.
DRAM_REFRESH	0x05	0-3	1	Number of DRAM Refreshes Issued
DRAM_PRE_ALL	0x06	0-3	1	DRAM Precharge All Commands
MAJOR_MODES	0x07	0-3	1	Cycles in a Major Mode
PREEMPTION	0x08	0-3	1	Read Preemption Count
ECC_CORRECTABLE_ERRORS	0x09	0-3	1	ECC Correctable Errors
RPQ_INSERTS	0x10	0-3	1	Read Pending Queue Allocations
RPQ_CYCLES_NE	0x11	0-3	1	Read Pending Queue Not Empty
WPQ_CYCLES_NE	0x21	0-3	1	Write Pending Queue Not Empty
WPQ_CYCLES_FULL	0x22	0-3	1	Write Pending Queue Full Cycles
WPQ_READ_HIT	0x23	0-3	1	Write Pending Queue CAM Match
WPQ_WRITE_HIT	0x24	0-3	1	Write Pending Queue CAM Match
POWER_THROTTLE_CYCLES	0x41	0-3	1	Throttle Cycles for Rank 0
POWER_PCU_THROTTLING	0x42	0-3	1	
POWER_SELF_REFRESH	0x43	0-3	0	Clock-Enabled Self-Refresh
POWER_CKE_CYCLES	0x83	0-3	16	CKE_ON_CYCLES by Rank
POWER_CHANNEL_DLLOFF	0x84	0-3	1	Channel DLLOFF Cycles
POWER_CHANNEL_PPD	0x85	0-3	4	Channel PPD Cycles
POWER_CRITICAL_THROTTLE_CYCLES	0x86	0-3	1	Critical Throttle Cycles
VMSE_WR_PUSH	0x90	0-3	1	VMSE WR PUSH issued
VMSE_MXB_WR_OCCUPANCY	0x91	0-3	32	VMSE MXB write buffer occupancy
RD_CAS_PRIO	0xa0	0-3	1	
BYP_CMDS	0xa1	0-3	1	
RD_CAS_RANK0	0xb0	0-3	1	RD_CAS Access to Rank 0
RD_CAS_RANK1	0xb1	0-3	1	RD_CAS Access to Rank 1
RD_CAS_RANK2	0xb2	0-3	1	RD_CAS Access to Rank 2
RD_CAS_RANK4	0xb4	0-3	1	RD_CAS Access to Rank 4
RD_CAS_RANK5	0xb5	0-3	1	RD_CAS Access to Rank 5
RD_CAS_RANK6	0xb6	0-3	1	RD_CAS Access to Rank 6
RD_CAS_RANK7	0xb7	0-3	1	RD_CAS Access to Rank 7
WR_CAS_RANK0	0xb8	0-3	1	WR_CAS Access to Rank 0
WR_CAS_RANK1	0xb9	0-3	1	WR_CAS Access to Rank 1
WR_CAS_RANK2	0xba	0-3	1	WR_CAS Access to Rank 2
WR_CAS_RANK3	0xbb	0-3	1	WR_CAS Access to Rank 3
WR_CAS_RANK4	0xbc	0-3	1	WR_CAS Access to Rank 4

Symbol Name	Event Code	Ctrs	Max Inc/Cyc	Description
WR_CAS_RANK5	0xbd	0-3	1	WR_CAS Access to Rank 5
WR_CAS_RANK6	0xbe	0-3	1	WR_CAS Access to Rank 6
WR_CAS_RANK7	0xbf	0-3	1	WR_CAS Access to Rank 7
WMM_TO_RMM	0xc0	0-3	1	Transition from WMM to RMM because of low threshold
WRONG_MM	0xc1	0-3	1	Not getting the requested Major Mode

2.5.6 iMC Box Common Metrics (Derived Events)

The following table summarizes metrics commonly calculated from iMC Box events.

Symbol Name: Definition	Equation
MEM_BW_READS: Memory bandwidth consumed by reads. Expressed in bytes.	(CAS_COUNT.RD * 64)
MEM_BW_TOTAL: Total memory bandwidth. Expressed in bytes.	MEM_BW_READS + MEM_BW_WRITES
MEM_BW_WRITES: Memory bandwidth consumed by writes Expressed in bytes.	(CAS_COUNT.WR * 64)
PCT_CYCLES_CRITICAL_THROTTLE: The percentage of cycles all DRAM ranks in critical thermal throttling	POWER_CRITICAL_THROTTLE_CYCLES / MC_Chy_PCI_PMON_CTR_FIXED
PCT_CYCLES_DLLOFF: The percentage of cycles all DRAM ranks in CKE slow (DLOFF) mode	POWER_CHANNEL_DLLOFF / MC_Chy_PCI_PMON_CTR_FIXED
PCT_CYCLES_DRAM_RANKx_IN_CKE: The percentage of cycles DRAM rank (x) spent in CKE ON mode.	POWER_CKE_CYCLES.RANKx / MC_Chy_PCI_PMON_CTR_FIXED
PCT_CYCLES_DRAM_RANKx_IN_THR: The percentage of cycles DRAM rank (x) spent in thermal throttling.	POWER_THROTTLE_CYCLES.RANKx / MC_Chy_PCI_PMON_CTR_FIXED
PCT_CYCLES_PPD: The percentage of cycles all DRAM ranks in PPD mode	POWER_CHANNEL_PPD / MC_Chy_PCI_PMON_CTR_FIXED
PCT_CYCLES_SELF_REFRESH: The percentage of cycles Memory is in self refresh power mode	POWER_SELF_REFRESH / MC_Chy_PCI_PMON_CTR_FIXED
PCT_RD_REQUESTS: Percentage of read requests from total requests.	RPQ_INSERTS / (RPQ_INSERTS + WPQ_INSERTS)
PCT_REQUESTS_PAGE_EMPTY: Percentage of memory requests that resulted in Page Empty	(ACT_COUNT - PRE_COUNT.PAGE_MISS)/ (CAS_COUNT.RD + CAS_COUNT.WR)
PCT_REQUESTS_PAGE_HIT: Percentage of memory requests that resulted in Page Hits	1 - (PCT_REQUESTS_PAGE_EMPTY + PCT_REQUESTS_PAGE_MISS)
PCT_REQUESTS_PAGE_MISS: Percentage of memory requests that resulted in Page Misses	PRE_COUNT.PAGE_MISS / (CAS_COUNT.RD + CAS_COUNT.WR)
PCT_WR_REQUESTS: Percentage of write requests from total requests.	WPQ_INSERTS / (RPQ_INSERTS + WPQ_INSERTS)

2.5.7 iMC Box Performance Monitor Event List

The section enumerates Intel® Xeon® Processor D-1500 Product Family performance monitoring events for the iMC Box.

ACT_COUNT

- **Title:** DRAM Activate Count
- **Category:** ACT Events
- **Event Code:** 0x01
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of DRAM Activate commands sent on this channel. Activate commands are issued to open up a page on the DRAM devices so that it can be read or written to with a CAS. One can calculate the number of Page Misses by subtracting the number of Page Miss precharges from the number of Activates.

Table 2-94. Unit Masks for ACT_COUNT

Extension	umask [15:8]	Description
RD	bxxxxxxx1	Activate due to Read
WR	bxxxxxx1x	Activate due to Write
BYP	bxxxx1xxx	Activate due to Write

BYP_CMDS

- **Title:**
- **Category:** BYPASS Command Events
- **Event Code:** 0xa1
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-95. Unit Masks for BYP_CMDS

Extension	umask [15:8]	Description
ACT	bxxxxxxx1	ACT command issued by 2 cycle bypass
CAS	bxxxxxx1x	CAS command issued by 2 cycle bypass
PRE	bxxxxx1xx	PRE command issued by 2 cycle bypass

CAS_COUNT

- **Title:** DRAM RD_CAS and WR_CAS Commands.
- **Category:** PRE Events
- **Event Code:** 0x04
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** DRAM RD_CAS and WR_CAS Commands

Table 2-96. Unit Masks for CAS_COUNT

Extension	umask [15:8]	Description
RD_REG	bxxxxxxx1	All DRAM RD_CAS (w/ and w/out auto-pre) Counts the total number or DRAM Read CAS commands issued on this channel. This includes both regular RD CAS commands as well as those with implicit Precharge. AutoPre is only used in systems that are using closed page policy. We do not filter based on major mode, as RD_CAS is not issued during WMM (with the exception of underfills).
RD_UNDERFILL	bxxxxxx1x	Underfill Read Issued Counts the number of underfill reads that are issued by the memory controller. This will generally be about the same as the number of partial writes, but may be slightly less because of partials hitting in the WPQ. While it is possible for underfills to be issed in both WMM and RMM, this event counts both.
RD	b00000011	All DRAM Reads (RD_CAS + Underfills) Counts the total number of DRAM Read CAS commands issued on this channel (including underfills).
WR_WMM	bxxxxx1xx	DRAM WR_CAS (w/ and w/out auto-pre) in Write Major Mode Counts the total number or DRAM Write CAS commands issued on this channel while in Write-Major-Mode.
WR_RMM	bxxxx1xxx	DRAM WR_CAS (w/ and w/out auto-pre) in Read Major Mode Counts the total number of Opportunistic" DRAM Write CAS commands issued on this channel while in Read-Major-Mode.
WR	b00001100	All DRAM WR_CAS (both Modes) Counts the total number of DRAM Write CAS commands issued on this channel.
ALL	b00001111	All DRAM WR_CAS (w/ and w/out auto-pre) Counts the total number of DRAM CAS commands issued on this channel.
RD_WMM	bxxx1xxxx	Read CAS issued in WMM
RD_RMM	bxx1xxxxx	Read CAS issued in RMM

DCLOCKTICKS

- **Title:** DRAM Clockticks
- **Category:** DCLK Events
- **Event Code:** 0x00
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

DRAM_PRE_ALL

- **Title:** DRAM Precharge All Commands
- **Category:** DRAM_PRE_ALL Events
- **Event Code:** 0x06
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times that the precharge all command was sent.

DRAM_REFRESH

- **Title:** Number of DRAM Refreshes Issued
- **Category:** DRAM_REFRESH Events
- **Event Code:** 0x05
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of refreshes issued.

Table 2-97. Unit Masks for DRAM_REFRESH

Extension	umask [15:8]	Description
PANIC	bxxxxxx1x	
HIGH	bxxxxx1xx	

ECC_CORRECTABLE_ERRORS

- **Title:** ECC Correctable Errors
- **Category:** ECC Events
- **Event Code:** 0x09
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of ECC errors detected and corrected by the iMC on this channel. This counter is only useful with ECC DRAM devices. This count will increment one time for each correction regardless of the number of bits corrected. The iMC can correct up to 4 bit errors in independent channel mode.

MAJOR_MODES

- **Title:** Cycles in a Major Mode
- **Category:** MAJOR_MODES Events
- **Event Code:** 0x07
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the total number of cycles spent in a major mode (selected by a filter) on the given channel. Major modea are channel-wide, and not a per-rank (or dimm or bank) mode.

Table 2-98. Unit Masks for MAJOR_MODES

Extension	umask [15:8]	Description
READ	bxxxxxxx1	Read Major Mode Read Major Mode is the default mode for the iMC, as reads are generally more critical to forward progress than writes.
WRITE	bxxxxxx1x	Write Major Mode This mode is triggered when the WPQ hits high occupancy and causes writes to be higher priority than reads. This can cause blips in the available read bandwidth in the system and temporarily increase read latencies in order to achieve better bus utilizations and higher bandwidth.
PARTIAL	bxxxxx1xx	Partial Major Mode This major mode is used to drain starved underfill reads. Regular reads and writes are blocked and only underfill reads will be processed.
ISOCH	bxxxx1xxx	Isoch Major Mode We group these two modes together so that we can use four counters to track each of the major modes at one time. These major modes are used whenever there is an ISOCH txn in the memory controller. In these mode, only ISOCH transactions are processed.

POWER_CHANNEL_DLLOFF

- **Title:** Channel DLLOFF Cycles
- **Category:** POWER Events
- **Event Code:** 0x84
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of cycles when all the ranks in the channel are in CKE Slow (DLLOFF) mode.
- **NOTE:** IBT = Input Buffer Termination = Off

POWER_CHANNEL_PPD

- **Title:** Channel PPD Cycles
- **Category:** POWER Events
- **Event Code:** 0x85
- **Max. Inc/Cyc:.** 4, **Register Restrictions:** 0-3
- **Definition:** Number of cycles when all the ranks in the channel are in PPD mode. If IBT=off is enabled, then this can be used to count those cycles. If it is not enabled, then this can count the number of cycles when that could have been taken advantage of.
- **NOTE:** IBT = Input Buffer Termination = On. ALL Ranks must be populated in order to measure

POWER_CKE_CYCLES

- **Title:** CKE_ON_CYCLES by Rank
- **Category:** POWER Events
- **Event Code:** 0x83
- **Max. Inc/Cyc:.** 16, **Register Restrictions:** 0-3
- **Definition:** Number of cycles spent in CKE ON mode. The filter allows you to select a rank to monitor. If multiple ranks are in CKE ON mode at one time, the counter will ONLY increment by one rather than doing accumulation. Multiple counters will need to be used to track multiple ranks simultaneously. There is no distinction between the different CKE modes (APD, PPDS, PPDF). This can be determined based on the system programming. These events should commonly be used with Invert to get the number of cycles in power saving mode. Edge Detect is also useful here. Make sure that you do NOT use Invert with Edge Detect (this just confuses the system and is not necessary).

Table 2-99. Unit Masks for POWER_CKE_CYCLES

Extension	umask [15:8]	Description
RANK0	b00000001	DIMM ID
RANK1	b00000010	DIMM ID
RANK2	b00000100	DIMM ID
RANK3	b00001000	DIMM ID
RANK4	b00010000	DIMM ID
RANK5	b00100000	DIMM ID
RANK6	b01000000	DIMM ID
RANK7	b10000000	DIMM ID

POWER_CRITICAL_THROTTLE_CYCLES

- **Title:** Critical Throttle Cycles
- **Category:** POWER Events
- **Event Code:** 0x86
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles when the iMC is in critical thermal throttling. When this happens, all traffic is blocked. This should be rare unless something bad is going on in the platform. There is no filtering by rank for this event.

POWER_PCU_THROTTLING

- **Title:**
- **Category:** POWER Events
- **Event Code:** 0x42
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

POWER_SELF_REFRESH

- **Title:** Clock-Enabled Self-Refresh
- **Category:** POWER Events
- **Event Code:** 0x43
- **Max. Inc/Cyc:.** 0, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles when the iMC is in self-refresh and the iMC still has a clock. This happens in some package C-states. For example, the PCU may ask the iMC to enter self-refresh even though some of the cores are still processing.

POWER_THROTTLE_CYCLES

- **Title:** Throttle Cycles for Rank 0
- **Category:** POWER Events
- **Event Code:** 0x41
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles while the iMC is being throttled by either thermal constraints or by the PCU throttling. It is not possible to distinguish between the two. This can be filtered by rank. If multiple ranks are selected and are being throttled at the same time, the counter will only increment by 1.

Table 2-100. Unit Masks for POWER_THROTTLE_CYCLES

Extension	umask [15:8]	Description
RANK0	bxxxxxxx1	DIMM ID Thermal throttling is performed per DIMM. We support 3 DIMMs per channel. This ID allows us to filter by ID.
RANK1	bxxxxxx1x	DIMM ID
RANK2	bxxxxx1xx	DIMM ID
RANK3	bxxxx1xxx	DIMM ID
RANK4	bxxx1xxxx	DIMM ID
RANK5	bxx1xxxxx	DIMM ID
RANK6	bx1xxxxxx	DIMM ID
RANK7	b1xxxxxxx	DIMM ID

PREEMPTION

- **Title:** Read Preemption Count
- **Category:** PREEMPTION Events
- **Event Code:** 0x08
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times a read in the iMC preempts another read or write. Generally reads to an open page are issued ahead of requests to closed pages. This improves the page hit rate of the system. However, high priority requests can cause pages of active requests to be closed in order to get them out. This will reduce the latency of the high-priority request at the expense of lower bandwidth and increased overall average latency.

Table 2-101. Unit Masks for PREEMPTION

Extension	umask [15:8]	Description
RD_PREEMPT_RD	bxxxxxxx1	Read over Read Preemption Filter for when a read preempts another read.
RD_PREEMPT_WR	bxxxxxx1x	Read over Write Preemption Filter for when a read preempts a write.

PRE_COUNT

- **Title:** DRAM Precharge commands.
- **Category:** PRE Events
- **Event Code:** 0x02
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of DRAM Precharge commands sent on this channel.

Table 2-102. Unit Masks for PRE_COUNT

Extension	umask [15:8]	Description
PAGE_MISS	bxxxxxxx1	Precharges due to page miss Counts the number of DRAM Precharge commands sent on this channel as a result of page misses. This does not include explicit precharge commands sent with CAS commands in Auto-Precharge mode. This does not include PRE commands sent as a result of the page close counter expiration.
PAGE_CLOSE	bxxxxxx1x	Precharge due to timer expiration Counts the number of DRAM Precharge commands sent on this channel as a result of the page close counter expiring. This does not include implicit precharge commands sent in auto-precharge mode.
RD	bxxxxx1xx	Precharge due to read
WR	bxxxx1xxx	Precharge due to write
BYP	bxxx1xxxx	Precharge due to bypass

RD_CAS_PRIO

- **Title:**
- **Category:** CAS Events
- **Event Code:** 0xa0
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-103. Unit Masks for RD_CAS_PRIO

Extension	umask [15:8]	Description
LOW	bxxxxxxx1	Read CAS issued with LOW priority
MED	bxxxxxx1x	Read CAS issued with MEDIUM priority
HIGH	bxxxxx1xx	Read CAS issued with HIGH priority
PANIC	bxxxx1xxx	Read CAS issued with PANIC NON ISOCH priority (starved)

RD_CAS_RANK0

- **Title:** RD_CAS Access to Rank 0
- **Category:** CAS Events
- **Event Code:** 0xb0
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-104. Unit Masks for RD_CAS_RANK0

Extension	umask [15:8]	Description
BANK0	b00000000	Bank 0
BANK1	b00000001	Bank 1
BANK2	b00000010	Bank 2
BANK3	b00000011	Bank 3
BANK4	b00000100	Bank 4
BANK5	b00000101	Bank 5
BANK6	b00000110	Bank 6
BANK7	b00000111	Bank 7
BANK8	b00001000	Bank 8
BANK9	b00001001	Bank 9
BANK10	b00001010	Bank 10
BANK11	b00001011	Bank 11
BANK12	b00001100	Bank 12
BANK13	b00001101	Bank 13
BANK14	b00001110	Bank 14
BANK15	b00001111	Bank 15
ALLBANKS	b00010000	All Banks
BANKG0	b00010001	Bank Group 0 (Banks 0-3)
BANKG1	b00010010	Bank Group 1 (Banks 4-7)
BANKG2	b00010011	Bank Group 2 (Banks 8-11)
BANKG3	b00010100	Bank Group 3 (Banks 12-15)

RD_CAS_RANK1

- **Title:** RD_CAS Access to Rank 1
- **Category:** CAS Events
- **Event Code:** 0xb1
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-105. Unit Masks for RD_CAS_RANK1

Extension	umask [15:8]	Description
BANK0	b00000000	Bank 0
BANK1	b00000001	Bank 1
BANK2	b00000010	Bank 2
BANK3	b00000011	Bank 3
BANK4	b00000100	Bank 4
BANK5	b00000101	Bank 5
BANK6	b00000110	Bank 6
BANK7	b00000111	Bank 7
BANK8	b00001000	Bank 8

Table 2-105. Unit Masks for RD_CAS_RANK1

Extension	umask [15:8]	Description
BANK9	b00001001	Bank 9
BANK10	b00001010	Bank 10
BANK11	b00001011	Bank 11
BANK12	b00001100	Bank 12
BANK13	b00001101	Bank 13
BANK14	b00001110	Bank 14
BANK15	b00001111	Bank 15
ALLBANKS	b00010000	All Banks
BANKG0	b00010001	Bank Group 0 (Banks 0-3)
BANKG1	b00010010	Bank Group 1 (Banks 4-7)
BANKG2	b00010011	Bank Group 2 (Banks 8-11)
BANKG3	b00010100	Bank Group 3 (Banks 12-15)

RD_CAS_RANK2

- **Title:** RD_CAS Access to Rank 2
- **Category:** CAS Events
- **Event Code:** 0xb2
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-106. Unit Masks for RD_CAS_RANK2

Extension	umask [15:8]	Description
BANK0	b00000000	Bank 0

RD_CAS_RANK4

- **Title:** RD_CAS Access to Rank 4
- **Category:** CAS Events
- **Event Code:** 0xb4
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-107. Unit Masks for RD_CAS_RANK4 (Sheet 1 of 2)

Extension	umask [15:8]	Description
BANK0	b00000000	Bank 0
BANK1	b00000001	Bank 1
BANK2	b00000010	Bank 2
BANK3	b00000011	Bank 3
BANK4	b00000100	Bank 4
BANK5	b00000101	Bank 5
BANK6	b00000110	Bank 6
BANK7	b00000111	Bank 7

Table 2-107. Unit Masks for RD_CAS_RANK4 (Sheet 2 of 2)

Extension	umask [15:8]	Description
BANK8	b00001000	Bank 8
BANK9	b00001001	Bank 9
BANK10	b00001010	Bank 10
BANK11	b00001011	Bank 11
BANK12	b00001100	Bank 12
BANK13	b00001101	Bank 13
BANK14	b00001110	Bank 14
BANK15	b00001111	Bank 15
ALLBANKS	b00010000	All Banks
BANKG0	b00010001	Bank Group 0 (Banks 0-3)
BANKG1	b00010010	Bank Group 1 (Banks 4-7)
BANKG2	b00010011	Bank Group 2 (Banks 8-11)
BANKG3	b00010100	Bank Group 3 (Banks 12-15)

RD_CAS_RANK5

- **Title:** RD_CAS Access to Rank 5
- **Category:** CAS Events
- **Event Code:** 0xb5
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-108. Unit Masks for RD_CAS_RANK5 (Sheet 1 of 2)

Extension	umask [15:8]	Description
BANK0	b00000000	Bank 0
BANK1	b00000001	Bank 1
BANK2	b00000010	Bank 2
BANK3	b00000011	Bank 3
BANK4	b00000100	Bank 4
BANK5	b00000101	Bank 5
BANK6	b00000110	Bank 6
BANK7	b00000111	Bank 7
BANK8	b00001000	Bank 8
BANK9	b00001001	Bank 9
BANK10	b00001010	Bank 10
BANK11	b00001011	Bank 11
BANK12	b00001100	Bank 12
BANK13	b00001101	Bank 13
BANK14	b00001110	Bank 14
BANK15	b00001111	Bank 15
ALLBANKS	b00010000	All Banks
BANKG0	b00010001	Bank Group 0 (Banks 0-3)

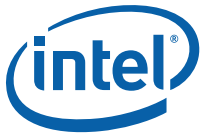


Table 2-108. Unit Masks for RD_CAS_RANK5 (Sheet 2 of 2)

Extension	umask [15:8]	Description
BANKG1	b00010010	Bank Group 1 (Banks 4-7)
BANKG2	b00010011	Bank Group 2 (Banks 8-11)
BANKG3	b00010100	Bank Group 3 (Banks 12-15)

RD_CAS_RANK6

- **Title:** RD_CAS Access to Rank 6
- **Category:** CAS Events
- **Event Code:** 0xb6
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-109. Unit Masks for RD_CAS_RANK6

Extension	umask [15:8]	Description
BANK0	b00000000	Bank 0
BANK1	b00000001	Bank 1
BANK2	b00000010	Bank 2
BANK3	b00000011	Bank 3
BANK4	b00000100	Bank 4
BANK5	b00000101	Bank 5
BANK6	b00000110	Bank 6
BANK7	b00000111	Bank 7
BANK8	b00001000	Bank 8
BANK9	b00001001	Bank 9
BANK10	b00001010	Bank 10
BANK11	b00001011	Bank 11
BANK12	b00001100	Bank 12
BANK13	b00001101	Bank 13
BANK14	b00001110	Bank 14
BANK15	b00001111	Bank 15
ALLBANKS	b00010000	All Banks
BANKG0	b00010001	Bank Group 0 (Banks 0-3)
BANKG1	b00010010	Bank Group 1 (Banks 4-7)
BANKG2	b00010011	Bank Group 2 (Banks 8-11)
BANKG3	b00010100	Bank Group 3 (Banks 12-15)

RD_CAS_RANK7

- **Title:** RD_CAS Access to Rank 7
- **Category:** CAS Events
- **Event Code:** 0xb7
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-110. Unit Masks for RD_CAS_RANK7

Extension	umask [15:8]	Description
BANK0	b00000000	Bank 0
BANK1	b00000001	Bank 1
BANK2	b00000010	Bank 2
BANK3	b00000011	Bank 3
BANK4	b00000100	Bank 4
BANK5	b00000101	Bank 5
BANK6	b00000110	Bank 6
BANK7	b00000111	Bank 7
BANK8	b00001000	Bank 8
BANK9	b00001001	Bank 9
BANK10	b00001010	Bank 10
BANK11	b00001011	Bank 11
BANK12	b00001100	Bank 12
BANK13	b00001101	Bank 13
BANK14	b00001110	Bank 14
BANK15	b00001111	Bank 15
ALLBANKS	b00010000	All Banks
BANKG0	b00010001	Bank Group 0 (Banks 0-3)
BANKG1	b00010010	Bank Group 1 (Banks 4-7)
BANKG2	b00010011	Bank Group 2 (Banks 8-11)
BANKG3	b00010100	Bank Group 3 (Banks 12-15)

RPQ_CYCLES_NE

- **Title:** Read Pending Queue Not Empty
- **Category:** RPQ Events
- **Event Code:** 0x11
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles that the Read Pending Queue is not empty. This can then be used to calculate the average occupancy (in conjunction with the Read Pending Queue Occupancy count). The RPQ is used to schedule reads out to the memory controller and to track the requests. Requests allocate into the RPQ soon after they enter the memory controller, and need credits for an entry in this buffer before being sent from the HA to the iMC. They deallocate after the CAS command has been issued to memory. This filter is to be used in conjunction with the occupancy filter so that one can correctly track the average occupancies for schedulable entries and scheduled requests.

RPQ_INSERTS

- **Title:** Read Pending Queue Allocations
- **Category:** RPQ Events
- **Event Code:** 0x10
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of allocations into the Read Pending Queue. This queue is used to schedule reads out to the memory controller and to track the requests. Requests allocate into the RPQ soon after they enter the memory controller, and need credits for an entry in this buffer before being sent from the HA to the iMC. They deallocate after the CAS command has been issued to memory. This includes both ISOCH and non-ISOCH requests.

VMSE_MXB_WR_OCCUPANCY

- **Title:** VMSE MXB write buffer occupancy
- **Category:** VMSE Events
- **Event Code:** 0x91
- **Max. Inc/Cyc:.** 32, **Register Restrictions:** 0-3
- **Definition:**

VMSE_WR_PUSH

- **Title:** VMSE WR PUSH issued
- **Category:** VMSE Events
- **Event Code:** 0x90
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-111. Unit Masks for VMSE_WR_PUSH

Extension	umask [15:8]	Description
WMM	bxxxxxxx1	VMSE write PUSH issued in WMM
RMM	bxxxxxx1x	VMSE write PUSH issued in RMM

WMM_TO_RMM

- **Title:** Transition from WMM to RMM because of low threshold
- **Category:** MAJOR_MODES Events
- **Event Code:** 0xc0
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-112. Unit Masks for WMM_TO_RMM

Extension	umask [15:8]	Description
LOW_THRESH	bxxxxxxx1	Transition from WMM to RMM because of starve counter
STARVE	bxxxxxx1x	
VMSE_RETRY	bxxxxx1xx	

WPQ_CYCLES_FULL

- **Title:** Write Pending Queue Full Cycles
- **Category:** WPQ Events
- **Event Code:** 0x22
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles when the Write Pending Queue is full. When the WPQ is full, the HA will not be able to issue any additional read requests into the iMC. This count should be similar count in the HA which tracks the number of cycles that the HA has no WPQ credits, just somewhat smaller to account for the credit return overhead.

WPQ_CYCLES_NE

- **Title:** Write Pending Queue Not Empty
- **Category:** WPQ Events
- **Event Code:** 0x21
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3

- **Definition:** Counts the number of cycles that the Write Pending Queue is not empty. This can then be used to calculate the average queue occupancy (in conjunction with the WPQ Occupancy Accumulation count). The WPQ is used to schedule write out to the memory controller and to track the writes. Requests allocate into the WPQ soon after they enter the memory controller, and need credits for an entry in this buffer before being sent from the HA to the iMC. They deallocate after being issued to DRAM. Write requests themselves are able to complete (from the perspective of the rest of the system) as soon they have "posted" to the iMC. This is not to be confused with actually performing the write to DRAM. Therefore, the average latency for this queue is actually not useful for deconstruction intermediate write latencies.

WPQ_READ_HIT

- **Title:** Write Pending Queue CAM Match
- **Category:** WPQ Events
- **Event Code:** 0x23
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times a request hits in the WPQ (write-pending queue). The iMC allows writes and reads to pass up other writes to different addresses. Before a read or a write is issued, it will first CAM the WPQ to see if there is a write pending to that address. When reads hit, they are able to directly pull their data from the WPQ instead of going to memory. Writes that hit will overwrite the existing data. Partial writes that hit will not need to do underfill reads and will simply update their relevant sections.

WPQ_WRITE_HIT

- **Title:** Write Pending Queue CAM Match
- **Category:** WPQ Events
- **Event Code:** 0x24
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times a request hits in the WPQ (write-pending queue). The iMC allows writes and reads to pass up other writes to different addresses. Before a read or a write is issued, it will first CAM the WPQ to see if there is a write pending to that address. When reads hit, they are able to directly pull their data from the WPQ instead of going to memory. Writes that hit will overwrite the existing data. Partial writes that hit will not need to do underfill reads and will simply update their relevant sections.

WRONG_MM

- **Title:** Not getting the requested Major Mode
- **Category:** MAJOR_MODES Events
- **Event Code:** 0xc1
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

WR_CAS_RANK0

- **Title:** WR_CAS Access to Rank 0
- **Category:** CAS Events
- **Event Code:** 0xb8
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-113. Unit Masks for WR_CAS_RANK0 (Sheet 1 of 2)

Extension	umask [15:8]	Description
BANK0	b00000000	Bank 0
BANK1	b00000001	Bank 1

Table 2-113. Unit Masks for WR_CAS_RANK0 (Sheet 2 of 2)

Extension	umask [15:8]	Description
BANK2	b00000010	Bank 2
BANK3	b00000011	Bank 3
BANK4	b00000100	Bank 4
BANK5	b00000101	Bank 5
BANK6	b00000110	Bank 6
BANK7	b00000111	Bank 7
BANK8	b00001000	Bank 8
BANK9	b00001001	Bank 9
BANK10	b00001010	Bank 10
BANK11	b00001011	Bank 11
BANK12	b00001100	Bank 12
BANK13	b00001101	Bank 13
BANK14	b00001110	Bank 14
BANK15	b00001111	Bank 15
ALLBANKS	b00010000	All Banks
BANKG0	b00010001	Bank Group 0 (Banks 0-3)
BANKG1	b00010010	Bank Group 1 (Banks 4-7)
BANKG2	b00010011	Bank Group 2 (Banks 8-11)
BANKG3	b00010100	Bank Group 3 (Banks 12-15)

WR_CAS_RANK1

- **Title:** WR_CAS Access to Rank 1
- **Category:** CAS Events
- **Event Code:** 0xb9
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-114. Unit Masks for WR_CAS_RANK1 (Sheet 1 of 2)

Extension	umask [15:8]	Description
BANK0	b00000000	Bank 0
BANK1	b00000001	Bank 1
BANK2	b00000010	Bank 2
BANK3	b00000011	Bank 3
BANK4	b00000100	Bank 4
BANK5	b00000101	Bank 5
BANK6	b00000110	Bank 6
BANK7	b00000111	Bank 7
BANK8	b00001000	Bank 8
BANK9	b00001001	Bank 9
BANK10	b00001010	Bank 10
BANK11	b00001011	Bank 11

Table 2-114. Unit Masks for WR_CAS_RANK1 (Sheet 2 of 2)

Extension	umask [15:8]	Description
BANK12	b00001100	Bank 12
BANK13	b00001101	Bank 13
BANK14	b00001110	Bank 14
BANK15	b00001111	Bank 15
ALLBANKS	b00010000	All Banks
BANKG0	b00010001	Bank Group 0 (Banks 0-3)
BANKG1	b00010010	Bank Group 1 (Banks 4-7)
BANKG2	b00010011	Bank Group 2 (Banks 8-11)
BANKG3	b00010100	Bank Group 3 (Banks 12-15)

WR_CAS_RANK2

- **Title:** WR_CAS Access to Rank 2
- **Category:** CAS Events
- **Event Code:** 0xba
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

WR_CAS_RANK3

- **Title:** WR_CAS Access to Rank 3
- **Category:** CAS Events
- **Event Code:** 0xbb
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

WR_CAS_RANK4

- **Title:** WR_CAS Access to Rank 4
- **Category:** CAS Events
- **Event Code:** 0xbc
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-117. Unit Masks for WR_CAS_RANK4 (Sheet 1 of 2)

Extension	umask [15:8]	Description
BANK0	b00000000	Bank 0
BANK1	b00000001	Bank 1
BANK2	b00000010	Bank 2
BANK3	b00000011	Bank 3
BANK4	b00000100	Bank 4
BANK5	b00000101	Bank 5
BANK6	b00000110	Bank 6
BANK7	b00000111	Bank 7
BANK8	b00001000	Bank 8

Table 2-117. Unit Masks for WR_CAS_RANK4 (Sheet 2 of 2)

Extension	umask [15:8]	Description
BANK9	b00001001	Bank 9
BANK10	b00001010	Bank 10
BANK11	b00001011	Bank 11
BANK12	b00001100	Bank 12
BANK13	b00001101	Bank 13
BANK14	b00001110	Bank 14
BANK15	b00001111	Bank 15
ALLBANKS	b00010000	All Banks
BANKG0	b00010001	Bank Group 0 (Banks 0-3)
BANKG1	b00010010	Bank Group 1 (Banks 4-7)
BANKG2	b00010011	Bank Group 2 (Banks 8-11)
BANKG3	b00010100	Bank Group 3 (Banks 12-15)

WR_CAS_RANK5

- **Title:** WR_CAS Access to Rank 5
- **Category:** CAS Events
- **Event Code:** 0xbd
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-118. Unit Masks for WR_CAS_RANK5 (Sheet 1 of 2)

Extension	umask [15:8]	Description
BANK0	b00000000	Bank 0
BANK1	b00000001	Bank 1
BANK2	b00000010	Bank 2
BANK3	b00000011	Bank 3
BANK4	b00000100	Bank 4
BANK5	b00000101	Bank 5
BANK6	b00000110	Bank 6
BANK7	b00000111	Bank 7
BANK8	b00001000	Bank 8
BANK9	b00001001	Bank 9
BANK10	b00001010	Bank 10
BANK11	b00001011	Bank 11
BANK12	b00001100	Bank 12
BANK13	b00001101	Bank 13
BANK14	b00001110	Bank 14
BANK15	b00001111	Bank 15
ALLBANKS	b00010000	All Banks
BANKG0	b00010001	Bank Group 0 (Banks 0-3)
BANKG1	b00010010	Bank Group 1 (Banks 4-7)

Table 2-118. Unit Masks for WR_CAS_RANK5 (Sheet 2 of 2)

Extension	umask [15:8]	Description
BANKG2	b00010011	Bank Group 2 (Banks 8-11)
BANKG3	b00010100	Bank Group 3 (Banks 12-15)

WR_CAS_RANK6

- **Title:** WR_CAS Access to Rank 6
- **Category:** CAS Events
- **Event Code:** 0xbe
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

Table 2-119. Unit Masks for WR_CAS_RANK6

Extension	umask [15:8]	Description
BANK0	b00000000	Bank 0
BANK1	b00000001	Bank 1
BANK2	b00000010	Bank 2
BANK3	b00000011	Bank 3
BANK4	b00000100	Bank 4
BANK5	b00000101	Bank 5
BANK6	b00000110	Bank 6
BANK7	b00000111	Bank 7
BANK8	b00001000	Bank 8
BANK9	b00001001	Bank 9
BANK10	b00001010	Bank 10
BANK11	b00001011	Bank 11
BANK12	b00001100	Bank 12
BANK13	b00001101	Bank 13
BANK14	b00001110	Bank 14
BANK15	b00001111	Bank 15
ALLBANKS	b00010000	All Banks
BANKG0	b00010001	Bank Group 0 (Banks 0-3)
BANKG1	b00010010	Bank Group 1 (Banks 4-7)
BANKG2	b00010011	Bank Group 2 (Banks 8-11)
BANKG3	b00010100	Bank Group 3 (Banks 12-15)

WR_CAS_RANK7

- **Title:** WR_CAS Access to Rank 7
- **Category:** CAS Events
- **Event Code:** 0xbf
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

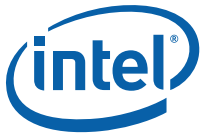


Table 2-120. Unit Masks for WR_CAS_RANK7

Extension	umask [15:8]	Description
BANK0	b00000000	Bank 0
BANK1	b00000001	Bank 1
BANK2	b00000010	Bank 2
BANK3	b00000011	Bank 3
BANK4	b00000100	Bank 4
BANK5	b00000101	Bank 5
BANK6	b00000110	Bank 6
BANK7	b00000111	Bank 7
BANK8	b00001000	Bank 8
BANK9	b00001001	Bank 9
BANK10	b00001010	Bank 10
BANK11	b00001011	Bank 11
BANK12	b00001100	Bank 12
BANK13	b00001101	Bank 13
BANK14	b00001110	Bank 14
BANK15	b00001111	Bank 15
ALLBANKS	b00010000	All Banks
BANKG0	b00010001	Bank Group 0 (Banks 0-3)
BANKG1	b00010010	Bank Group 1 (Banks 4-7)
BANKG2	b00010011	Bank Group 2 (Banks 8-11)
BANKG3	b00010100	Bank Group 3 (Banks 12-15)

2.6 IRP Performance Monitoring

IRP is responsible for maintaining coherency for IIO traffic that needs to be coherent (e.g. cross-socket P2P)

2.6.1 IRP Performance Monitoring Overview

The IRP Box supports event monitoring through two sets of two 48b wide counters (IRP{0,1}_PCI_PMON_CTR/CTL{1:0}). Each of these four counters can be programmed to count any IRP event. The IRP counters can increment by a maximum of 7b per cycle.

For information on how to setup a monitoring session, refer to Section 2.1, "Uncore Per-Socket Performance Monitoring Control".

2.6.1.1 IRP PMON Registers - On Overflow and the Consequences (PMI/Freeze)

If an overflow is detected from an IRP performance counter, the overflow bit is set at the box level (IRP_PCI_PMON_BOX_STATUS.ov). If the counter is enabled to communicate the overflow (IRP_PCI_PMON_CTL.ov_en is set to 1), an overflow message is sent to the UBox. When the UBox receives the overflow signal, the U_MSR_PMON_GLOBAL_STATUS.ov_i bit is set (see Table 2-3, "U_MSR_PMON_GLOBAL_STATUS Register – Field Definitions"), a global freeze signal is sent and a PMI can be generated.

Once a freeze has occurred, in order to see a new freeze, the overflow responsible for the freeze must be cleared by setting the corresponding bit in IRP_PCI_PMON_BOX_STATUS.ov and U_MSR_PMON_GLOBAL_STATUS.ov_i to 1 (which acts to clear the bits). Assuming all the counters have been locally enabled (.*en* bit in control registers meant to monitor events) and the overflow bits have been cleared, the IRP is prepared for a new sample interval. Once the global controls have been re-enabled (Section 2.1.4, "Enabling a New Sample Interval from Frozen Counters"), counting will resume.

2.6.2 IRP Performance Monitors

Table 2-121. IRP Performance Monitoring Registers (PCICFG) (Sheet 1 of 2)

Register Name	PCICFG Address	Size (bits)	Description
PCICFG Base Address	Dev:Func DeviceID		
IRP PMON Registers	D5:F6 0x6F39		
Box-Level Control/Status			
IRP_PCI_PMON_BOX_STATUS	F8	32	IRP PMON Box-Wide Status
IRP_PCI_PMON_BOX_CTL	F4	32	IRP PMON Box-Wide Control
Generic Counter Control			
IRP1_PCI_PMON_CTL1	E4	32	IRP 1 PMON Control for Counter 1
IRP1_PCI_PMON_CTL0	E0	32	IRP 1 PMON Control for Counter 0
IRP0_PCI_PMON_CTL1	DC	32	IRP 0 PMON Control for Counter 1

Table 2-121. IRP Performance Monitoring Registers (PCICFG) (Sheet 2 of 2)

Register Name	PCICFG Address	Size (bits)	Description
IRP0_PCI_PMON_CTL0	D8	32	IRP 0 PMON Control for Counter 0
Generic Counters			
IRP1_PCI_PMON_CTR1	C0	64	IRP 1 PMON Counter 1
IRP1_PCI_PMON_CTR0	B8	64	IRP 1 PMON Counter 0
IRP0_PCI_PMON_CTR1	B0	64	IRP 0 PMON Counter 1
IRP0_PCI_PMON_CTR0	A0	64	IRP 0 PMON Counter 0

2.6.2.1 IRP Box Level PMON State

The following registers represent the state governing all box-level PMUs in the IRP Box.

In the case of the IRP, the IRP_PCI_PMON_BOX_CTL register provides the ability to manually freeze the counters in the box (*.frz*) and reset the generic state (*.rst_ctrs* and *.rst_ctrl*).

If an overflow is detected from one of the IRP PMON registers, the corresponding bit in the IRP_PCI_PMON_BOX_STATUS.ov field will be set. To reset these overflow bits, a user must write a value of '1' to them (which will clear the bits).

Table 2-122. IRP_PCI_PMON_BOX_CTL Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
rsv	31:18	RV	0	Ignored
rsv	17:16	RV	0	Reserved; SW must write to 1 else behavior is undefined.
rsv	15:9	RV	0	Ignored
frz	8	WO	0	Freeze. If set to 1 the counters in this box will be frozen.
ig	7:2	RV	0	Ignored
rst_ctrs	1	WO	0	Reset Counters. When set to 1, the Counter Registers will be reset to 0.
rst_ctrl	0	WO	0	Reset Control. When set to 1, the Counter Control Registers will be reset to 0.

Table 2-123. IRP_PCI_PMON_BOX_STATUS Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
ig	31:4	RV	0	Ignored
ov	3:0	RW1C	0	If an overflow is detected from the corresponding IRP_PCI_PMON_CTR register, it's overflow bit will be set. NOTE: Write of '1' will clear the bit.

2.6.2.2 IRP PMON state - Counter/Control Pairs

The following table defines the layout of the IRP performance monitor control registers. The main task of these configuration registers is to select the event to be monitored by their respective data counter (*.ev_sel*, *.umask*). Additional control bits are provided to shape the incoming events (e.g. *.invert*, *.edge_det*, *.thresh*) as well as provide additional functionality for monitoring software (*.rst*,*.ov_en*).

Table 2-124. IRP_PCI_PMON_CTL{3-0} Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
thresh	31:24	RW-V	0	Threshold used in counter comparison.
invert	23	RW-V	0	Invert comparison against Threshold. 0 - comparison will be 'is event increment >= threshold?'. 1 - comparison is inverted - 'is event increment < threshold?' NOTE: .invert is in series following .thresh, Due to this, the .thresh field must be set to a non-0 value. For events that increment by no more than 1 per cycle, set .thresh to 0x1. Also, if .edge_det is set to 1, the counter will increment when a 1 to 0 transition (i.e. falling edge) is detected.
en	22	RW-V	0	Local Counter Enable.
rsv	21:20	RV	0	Reserved. SW must write to 0 else behavior is undefined.
ov_en	20	RW-V	0	When this bit is asserted and the corresponding counter overflows, its overflow bit is set in the local status register (IRP_PCI_PMON_BOX_STATUS.ov) and an overflow is sent on the message channel to the UBox. When the overflow is received by the UBox, the bit corresponding to this IRP will be set in U_MSR_PMON_GLOBAL_STATUS.ov_i
ig	19	RV	0	Ignored
edge_det	18	RW-V	0	When set to 1, rather than measuring the event in each cycle it is active, the corresponding counter will increment when a 0 to 1 transition (i.e. rising edge) is detected. When 0, the counter will increment in each cycle that the event is asserted. NOTE: .edge_det is in series following .thresh, Due to this, the .thresh field must be set to a non-0 value. For events that increment by no more than 1 per cycle, set .thresh to 0x1.
rst	17	WO	0	When set to 1, the corresponding counter will be cleared to 0.
rsv	16	RV	0	Reserved. SW must write to 0 else behavior is undefined.
umask	15:8	RW-V	0	Select subevents to be counted within the selected event.
ev_sel	7:0	RW-V	0	Select event to be counted.

The IRP performance monitor data registers are 48-bit wide. A counter overflow occurs when a carry out from bit 47 is detected. Software can force all uncore counting to freeze after N events by preloading a monitor with a count value of 2^{48} - N and setting the control register to send an overflow message to the UBox (Section 2.1.1.1, "Freezing on Counter Overflow"). During the interval of time between overflow and global disable, the counter value will wrap and continue to collect events.

If accessible, software can continuously read the data registers without disabling event collection.

Table 2-125. IRP{0,1}_PCI_PMON_CTR{1-0} Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
ig	63:48	RV	0	Ignored
event_count	47:0	RW-V	0	44-bit performance event counter

2.6.3 IRP Performance Monitoring Events

IRP provides events to track information related to all the traffic passing through it’s boundaries.

- Write Cache Occupancy
- Ingress/Egress Traffic - by Ring Type
- Stalls awaiting Credits

2.6.4 IRP Box Events Ordered By Code

The following table summarizes the directly measured IRP Box events.

Symbol Name	Event Code	Ctrs	Max Inc/Cyc	Description
CLOCKTICKS	0x00	0-1	1	Clocks in the IRP
RxR_BL_DRS_INSERTS	0x01	0-1	1	BL Ingress Occupancy - DRS
RxR_BL_NCB_INSERTS	0x02	0-1	1	BL Ingress Occupancy - NCB
RxR_BL_NCS_INSERTS	0x03	0-1	1	BL Ingress Occupancy - NCS
RxR_BL_DRS_CYCLES_FULL	0x04	0-1	1	
RxR_BL_NCB_CYCLES_FULL	0x05	0-1	1	
RxR_BL_NCS_CYCLES_FULL	0x06	0-1	1	
RxR_BL_DRS_OCCUPANCY	0x07	0-1	24	
RxR_BL_NCB_OCCUPANCY	0x08	0-1	24	
RxR_BL_NCS_OCCUPANCY	0x09	0-1	24	
RxR_AK_INSERTS	0x0a	0-1	1	AK Ingress Occupancy
TxR_REQUEST_OCCUPANCY	0x0d	0-1	1	Outbound Request Queue Occupancy
TxR_DATA_INSERTS_NCB	0x0e	0-1	1	Outbound Read Requests
TxR_DATA_INSERTS_NCS	0x0f	0-1	1	Outbound Read Requests
CACHE_TOTAL_OCCUPANCY	0x12	0-1	128	Total Write Cache Occupancy
COHERENT_OPS	0x13	0-1	1	Coherent Ops
MISC0	0x14	0-1	1	Misc Events - Set 0
MISC1	0x15	0-1	1	Misc Events - Set 1
TRANSACTIONS	0x16	0-1	1	Inbound Transaction Count
SNOOP_RESP	0x17	0-1	1	Snoop Responses
TxR_AD_STALL_CREDIT_CYCLES	0x18	0-1	1	No AD Egress Credit Stalls
TxR_BL_STALL_CREDIT_CYCLES	0x19	0-1	1	No BL Egress Credit Stalls

2.6.5 IRP Box Performance Monitor Event List

The section enumerates Intel® Xeon® Processor D-1500 Product Family performance monitoring events for the IRP Box.

CACHE_TOTAL_OCCUPANCY

- **Title:** Total Write Cache Occupancy
- **Category:** WRITE_CACHE Events
- **Event Code:** 0x12
- **Max. Inc/Cyc:.** 128, **Register Restrictions:** 0-1
- **Definition:** Accumulates the number of reads and writes that are outstanding in the uncore in each cycle. This is effectively the sum of the READ_OCCUPANCY and WRITE_OCCUPANCY events.

Table 2-126. Unit Masks for CACHE_TOTAL_OCCUPANCY

Extension	umask [15:8]	Description
ANY	b00000001	Any Source Tracks all requests from any source port.
SOURCE	b00000010	Select Source Tracks only those requests that come from the port specified in the IRP_PmonFilter.OrderingQ register. This register allows one to select one specific queue. It is not possible to monitor multiple queues at a time.

CLOCKTICKS

- **Title:** Clocks in the IRP
- **Category:** IO_CLKS Events
- **Event Code:** 0x00
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** Number of clocks in the IRP.

COHERENT_OPS

- **Title:** Coherent Ops
- **Category:** Coherency Events
- **Event Code:** 0x13
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** Counts the number of coherency related operations servied by the IRP

Table 2-127. Unit Masks for COHERENT_OPS

Extension	umask [15:8]	Description
PCIRDCUR	bxxxxxxx1	PCIRdCur
CRD	bxxxxxx1x	CRd
DRD	bxxxxx1xx	DRd
RFO	bxxxx1xxx	RFO
PCITOM	bxxx1xxxx	PCIItoM
PCIDCAHINT	bxx1xxxxx	PCIDCAHin5t
WBMTOI	bx1xxxxxx	WbMtoI
CLFLUSH	b1xxxxxxx	CLFlush

MISC0

- **Title:** Misc Events - Set 0
- **Category:** MISC Events
- **Event Code:** 0x14
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:**

Table 2-128. Unit Masks for MISC0

Extension	umask [15:8]	Description
FAST_REQ	b000000x1	Fastpath Requests
FAST_REJ	b0000001x	Fastpath Rejects
2ND_RD_INSERT	bx00xx100	Cache Inserts of Read Transactions as Secondary
2ND_WR_INSERT	bx00x1x00	Cache Inserts of Write Transactions as Secondary
2ND_ATOMIC_INSERT	bx001xx00	Cache Inserts of Atomic Transactions as Secondary
FAST_XFER	bxx100000	Fastpath Transfers From Primary to Secondary
PF_ACK_HINT	bx1x00000	Prefetch Ack Hints From Primary to Secondary
PF_TIMEOUT	b1xx00000	Prefetch TimeOut Indicates the fetch for a previous prefetch wasn't accepted by the prefetch. This happens in the case of a prefetch TimeOut

MISC1

- **Title:** Misc Events - Set 1
- **Category:** MISC Events
- **Event Code:** 0x15
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:**

Table 2-129. Unit Masks for MISC1

Extension	umask [15:8]	Description
SLOW_I	b000xxxx1	Slow Transfer of I Line Snoop took cacheline ownership before write from data was committed.
SLOW_S	b000xxx1x	Slow Transfer of S Line Secondary received a transfer that did not have sufficient MESI state
SLOW_E	b000xx1xx	Slow Transfer of E Line Secondary received a transfer that did have sufficient MESI state
SLOW_M	b000x1xxx	Slow Transfer of M Line Snoop took cacheline ownership before write from data was committed.
LOST_FWD	b0001xxxx	
SEC_RCVD_INVLD	bxx1x0000	Received Invalid Secondary received a transfer that did not have sufficient MESI state
SEC_RCVD_VLD	bx1xx0000	Received Valid Secondary received a transfer that did have sufficient MESI state
DATA_THROTTLE	b1xxx0000	Data Throttled IRP throttled switch data

RxR_AK_INSERTS

- **Title:** AK Ingress Occupancy
- **Category:** AK_INGRESS Events
- **Event Code:** 0x0a
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** Counts the number of allocations into the AK Ingress. This queue is where the IRP receives responses from R2PCIe (the ring).

RxR_BL_DRS_CYCLES_FULL

- **Title:**
- **Category:** BL_INGRESS_DRS Events
- **Event Code:** 0x04
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** Counts the number of cycles when the BL Ingress is full. This queue is where the IRP receives data from R2PCIe (the ring). It is used for data returns from read requests as well as outbound MMIO writes.

RxR_BL_DRS_INSERTS

- **Title:** BL Ingress Occupancy - DRS
- **Category:** BL_INGRESS_DRS Events
- **Event Code:** 0x01
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** Counts the number of allocations into the BL Ingress. This queue is where the IRP receives data from R2PCIe (the ring). It is used for data returns from read requests as well as outbound MMIO writes.

RxR_BL_DRS_OCCUPANCY

- **Title:**
- **Category:** BL_INGRESS_DRS Events
- **Event Code:** 0x07
- **Max. Inc/Cyc:.** 24, **Register Restrictions:** 0-1
- **Definition:** Accumulates the occupancy of the BL Ingress in each cycles. This queue is where the IRP receives data from R2PCIe (the ring). It is used for data returns from read requests as well as outbound MMIO writes.

RxR_BL_NCB_CYCLES_FULL

- **Title:**
- **Category:** BL_INGRESS_NCB Events
- **Event Code:** 0x05
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** Counts the number of cycles when the BL Ingress is full. This queue is where the IRP receives data from R2PCIe (the ring). It is used for data returns from read request as well as outbound MMIO writes.

RxR_BL_NCB_INSERTS

- **Title:** BL Ingress Occupancy - NCB
- **Category:** BL_INGRESS_NCB Events
- **Event Code:** 0x02
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** Counts the number of allocations into the BL Ingress. This queue is where the IRP receives data from R2PCIe (the ring). It is used for data returns from read request as well as outbound MMIO writes.

RxR_BL_NCB_OCCUPANCY

- **Title:**
- **Category:** BL_INGRESS_NCB Events
- **Event Code:** 0x08
- **Max. Inc/Cyc:.** 24, **Register Restrictions:** 0-1

- **Definition:** Accumulates the occupancy of the BL Ingress in each cycles. This queue is where the IRP receives data from R2PCIe (the ring). It is used for data returns from read request as well as outbound MMIO writes.

RxR_BL_NCS_CYCLES_FULL

- **Title:**
- **Category:** BL_INGRESS_NCS Events
- **Event Code:** 0x06
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** Counts the number of cycles when the BL Ingress is full. This queue is where the IRP receives data from R2PCIe (the ring). It is used for data returns from read request as well as outbound MMIO writes.

RxR_BL_NCS_INSERTS

- **Title:** BL Ingress Occupancy - NCS
- **Category:** BL_INGRESS_NCS Events
- **Event Code:** 0x03
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** Counts the number of allocations into the BL Ingress. This queue is where the IRP receives data from R2PCIe (the ring). It is used for data returns from read request as well as outbound MMIO writes.

RxR_BL_NCS_OCCUPANCY

- **Title:**
- **Category:** BL_INGRESS_NCS Events
- **Event Code:** 0x09
- **Max. Inc/Cyc:.** 24, **Register Restrictions:** 0-1
- **Definition:** Accumulates the occupancy of the BL Ingress in each cycles. This queue is where the IRP receives data from R2PCIe (the ring). It is used for data returns from read request as well as outbound MMIO writes.

SNOOP_RESP

- **Title:** Snoop Responses
- **Category:** TRANSACTIONS Events
- **Event Code:** 0x17
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:**
- **NOTE:** The first 4 subevent bits are the Responses to the Code/Data/Invalid Snoops represented by the last 3 subevent bits. At least 1 of the bottom 4 bits must be combined with 1 of the top 3 bits to obtain counts. Unsure which combinations are possible.

Table 2-130. Unit Masks for SNOOP_RESP

Extension	umask [15:8]	Description
MISS	bxxxxxxx1	Miss
HIT_I	bxxxxxx1x	Hit I
HIT_ES	bxxxxx1xx	Hit E or S
HIT_M	bxxxx1xxx	Hit M
SNPCODE	bxxx1xxxx	SnpCode
SNPDATA	bxx1xxxxx	SnpData
SNPINV	bx1xxxxxx	SnpInv

TRANSACTIONS

- **Title:** Inbound Transaction Count
- **Category:** TRANSACTIONS Events
- **Event Code:** 0x16
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** Counts the number of "Inbound" transactions from the IRP to the Uncore. This can be filtered based on request type in addition to the source queue. Note the special filtering equation. We do OR-reduction on the request type. If the SOURCE bit is set, then we also do AND qualification based on the source portID.
- **NOTE:** Bit 7 is a filter that can be applied to the other subevents. Meaningless by itself.

Table 2-131. Unit Masks for TRANSACTIONS

Extension	umask [15:8]	Filter Dep	Description
READS	bxxxxxxx1		Reads Tracks only read requests (not including read prefetches).
WRITES	bxxxxxx1x		Writes Tracks only write requests. Each write request should have a prefetch, so there is no need to explicitly track these requests. For writes that are tickled and have to retry, the counter will be incremented for each retry.
RD_PREF	bxxxxx1xx		Read Prefetches Tracks the number of read prefetches.
WR_PREF	bxxxx1xxx		Write Prefetches Tracks the number of write prefetches.
ATOMIC	bxxx1xxxx		Atomic Tracks the number of atomic transactions
OTHER	bxx1xxxxx		Other Tracks the number of 'other' kinds of transactions.
ORDERINGQ	bx1xxxxxx	IRPFilter[4:0]	Select Source Tracks only those requests that come from the port specified in the IRP_PmonFilter.OrderingQ register. This register allows one to select one specific queue. It is not possible to monitor multiple queues at a time. If this bit is not set, then requests from all sources will be counted.

TxR_AD_STALL_CREDIT_CYCLES

- **Title:** No AD Egress Credit Stalls
- **Category:** STALL_CYCLES Events
- **Event Code:** 0x18
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** Counts the number times when it is not possible to issue a request to the R2PCIe because there are no AD Egress Credits available.

TxR_BL_STALL_CREDIT_CYCLES

- **Title:** No BL Egress Credit Stalls
- **Category:** STALL_CYCLES Events
- **Event Code:** 0x19
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** Counts the number times when it is not possible to issue data to the R2PCIe because there are no BL Egress Credits available.

TxR_DATA_INSERTS_NCB

- **Title:** Outbound Read Requests
- **Category:** OUTBOUND_REQUESTS Events
- **Event Code:** 0x0e
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** Counts the number of requests issued to the switch (towards the devices).

TxR_DATA_INSERTS_NCS

- **Title:** Outbound Read Requests
- **Category:** OUTBOUND_REQUESTS Events
- **Event Code:** 0x0f
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** Counts the number of requests issued to the switch (towards the devices).

TxR_REQUEST_OCCUPANCY

- **Title:** Outbound Request Queue Occupancy
- **Category:** OUTBOUND_REQUESTS Events
- **Event Code:** 0x0d
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** Accumultes the number of outstanding outbound requests from the IRP to the switch (towards the devices). This can be used in conjunction with the allocations event in order to calculate average latency of outbound requests.

2.7 Power Control (PCU) Performance Monitoring

The PCU is the primary Power Controller for the Intel® Xeon® Processor D-1500 Product Family

The uncore implements a power control unit acting as a core/uncore power and thermal manager. It runs its firmware on an internal micro-controller and coordinates the socket's power states.

The PCU algorithmically governs the P-state of the processor, C-state of the core and the package C-state of the socket. It also enables the core to go to a higher performance state ("turbo mode") when the proper set of conditions are met. Conversely, the PCU will throttle the processor to a lower performance state when a thermal violation occurs.

Through specific events, the OS and the PCU will either promote or demote the C-State of each core by altering the voltage and frequency. The system power state (S-state) of all the sockets in the system is managed by the server legacy bridge in coordination with all socket PCUs.

The PCU communicates to all the other units through multiple PMLink interfaces on-die and Message Channel to access their registers. The OS and BIOS communicates to the PCU thru standardized MSR registers and ACPI.

The PCU also acts as the interface to external management controllers via PECI and voltage regulators (NPTM). The DMI2 interface is the communication path from the southbridge for system power management.

Note: Many power saving features are tracked as events in their respective units. For example, Intel® QPI Link Power saving states and Memory CKE statistics are captured in the Intel® QPI Perfmon and IMC Perfmon respectively.

2.7.1 PCU Performance Monitoring Overview

The uncore PCU supports event monitoring through four 48-bit wide counters (PCU_MSR_PMON_CTR{3:0}). Each of these counters can be programmed (PCU_MSR_PMON_CTL{3:0}) to monitor any PCU event. The PCU counters can increment by a maximum of 5b per cycle.

Two extra 64-bit counters are also provided by the Intel® Xeon® Processor D-1500 Product FamilyPCU to track C-State Residence. Although documented in this manual for reference, these counters exist outside of the PMON infrastructure.

For information on how to setup a monitoring session, refer to Section 2.1, "Uncore Per-Socket Performance Monitoring Control".

2.7.1.1 PCU PMON Registers - On Overflow and the Consequences (PMI/Freeze)

If an overflow is detected from an PCU performance counter, the overflow bit is set at the box level (PCU_MSR_PMON_BOX_STATUS.ov). If the counter is enabled to communicate the overflow (PCU_MSR_PMON_CTL.ov_en is set to 1), an overflow message is sent to the UBox. When the UBox receives the overflow signal, the U_MSR_PMON_GLOBAL_STATUS.ov_p bit is set (see Table 2-3, "U_MSR_PMON_GLOBAL_STATUS Register – Field Definitions"), a global freeze signal is sent and a PMI can be generated.

Once a freeze has occurred, in order to see a new freeze, the overflow responsible for the freeze must be cleared by setting the corresponding bit in PCU_MSR_PMON_BOX_STATUS.ov and U_MSR_PMON_GLOBAL_STATUS.ov_p to 1 (which acts to clear the bits). Assuming all the counters have been locally enabled (*.en* bit in control registers meant to monitor events) and the overflow bits

have been cleared, the PCU is prepared for a new sample interval. Once the global controls have been re-enabled (Section 2.1.4, “Enabling a New Sample Interval from Frozen Counters”), counting will resume.

2.7.2 PCU Performance Monitors

Table 2-132. PCU Performance Monitoring Registers (MSR)

MSR Name	MSR Address	Size (bits)	Description
Generic Counters			
PCU_MSR_PMON_CTR3	0x071A	64	PCU PMON Counter 3
PCU_MSR_PMON_CTR2	0x0719	64	PCU PMON Counter 2
PCU_MSR_PMON_CTR1	0x0718	64	PCU PMON Counter 1
PCU_MSR_PMON_CTR0	0x0717	64	PCU PMON Counter 0
Box-Level Filter			
PCU_MSR_PMON_BOX_FILTER	0x0715	32	PCU PMON Filter
Generic Counter Control			
PCU_MSR_PMON_CTL3	0x0714	32	PCU PMON Control for Counter 3
PCU_MSR_PMON_CTL2	0x0713	32	PCU PMON Control for Counter 2
PCU_MSR_PMON_CTL1	0x0712	32	PCU PMON Control for Counter 1
PCU_MSR_PMON_CTL0	0x0711	32	PCU PMON Control for Counter 0
Box-Level Control/Status			
PCU_MSR_PMON_BOX_STATUS	0x0716	32	PCU PMON Box-Wide Status
PCU_MSR_PMON_BOX_CTL	0x0710	32	PCU PMON Box-Wide Control
Fixed (Non-PMON) Counters			
PCU_MSR_CORE_C6_CTR	0x03FD	64	Fixed C-State Residency Counter
PCU_MSR_CORE_C3_CTR	0x03FC	64	Fixed C-State Residency Counter

2.7.2.1 PCU Box Level PMON State

The following registers represent the state governing all box-level PMUs in the PCU.

In the case of the PCU, the PCU_MSR_PMON_BOX_CTL register provides the ability to manually freeze the counters in the box (*.frz*) and reset the generic state (*.rst_ctrs* and *.rst_ctrl*).

The PCU provides two extra MSRs that provide additional static performance information to software but exist outside of the PMON infrastructure (e.g. they can’t be frozen or reset). They are included for the convenience of software developers need to efficiently access this data.

If an overflow is detected from one of the PCU PMON registers, the corresponding bit in the PCU_MSR_PMON_BOX_STATUS.ov field will be set. To reset these overflow bits, a user must write a value of ‘1’ to them (which will clear the bits).

Table 2-133. PCU_MSR_PMON_BOX_CTL Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
rsv	31:18	RV	0	Reserved
rsv	17:16	RV	0	Reserved; SW must write to 1 else behavior is undefined.
rsv	15:9	RV	0	Reserved
frz	8	WO	0	Freeze. If set to 1 the counters in this box will be frozen.
rsv	7:2	RV	0	Reserved
rst_ctrs	1	WO	0	Reset Counters. When set to 1, the Counter Registers will be reset to 0.
rst_ctrl	0	WO	0	Reset Control. When set to 1, the Counter Control Registers will be reset to 0.

Table 2-134. PCU_MSR_PMON_BOX_STATUS Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
rsv	31:4	RV	0	Reserved
ov	3:0	RW1C	0	If an overflow is detected from the corresponding PCU_MSR_PMON_CTR register, it's overflow bit will be set. NOTE: Write of '1' will clear the bit.

2.7.2.2 PCU PMON state - Counter/Control Pairs

The following table defines the layout of the PCU performance monitor control registers. The main task of these configuration registers is to select the event to be monitored by their respective data counter (*.ev_sel*, *.umask*). Additional control bits are provided to shape the incoming events (e.g. *.invert*, *.edge_det*, *.thresh*) as well as provide additional functionality for monitoring software (*.rst*,*.ov_en*).

Due to the fact that much of the PCU's functionality is provided by an embedded microcontroller, many of the available events are generated by the microcontroller and handed off to the hardware for capture by the PMON registers. Among the events generated by the microcontroller are occupancy events allowing a user to measure the number of cores in a given C-state per-cycle. Given this unique situation, extra control bits are provided to filter the output of the these special occupancy events.

- *.occ_invert* - Changes the *.thresh* test condition to '<' for the occupancy events (when .ev_sel[7] is set to 1)

- *.occ_edge_det* - Rather than accumulating the raw count each cycle (for events that can increment by 1 per cycle), the register can capture transitions from no event to an event incoming for the PCU's occupancy events (when .ev_sel[7] is set to 1).

Table 2-135. PCU_MSR_PMON_CTL{3-0} Register – Field Definitions (Sheet 1 of 2)

Field	Bits	Attr	HW Reset Val	Description
occ_edge_det	31	RW-V	0	Enables edge detect for occupancy events (.ev_sel[7] is 1) When set to 1, rather than measuring the event in each cycle it is active, the corresponding counter will increment when a 0 to 1 transition (i.e. rising edge) is detected. When 0, the counter will increment in each cycle that the event is asserted. NOTE: .edge_det is in series following .thresh, Due to this, the .thresh field must be set to a non-0 value. For events that increment by no more than 1 per cycle, set .thresh to 0x1.
occ_invert	30	RW-V	0	Invert comparison against Threshold for the PCU Occupancy events (.ev_sel[7] is 1) 0 - comparison will be 'is event increment >= threshold?'. 1 - comparison is inverted - 'is event increment < threshold?' NOTE: .invert is in series following .thresh, Due to this, the .thresh field must be set to a non-0 value. For events that increment by no more than 1 per cycle, set .thresh to 0x1. Also, if .edge_det is set to 1, the counter will increment when a 1 to 0 transition (i.e. falling edge) is detected.
rsv	29	RV	0	Reserved. SW must write to 0 else behavior is undefined.
thresh	28:24	RW-V	0	Threshold used in counter comparison.
invert	23	RW-V	0	Invert comparison against Threshold. 0 - comparison will be 'is event increment >= threshold?'. 1 - comparison is inverted - 'is event increment < threshold?' NOTE: .invert is in series following .thresh, Due to this, the .thresh field must be set to a non-0 value. For events that increment by no more than 1 per cycle, set .thresh to 0x1. Also, if .edge_det is set to 1, the counter will increment when a 1 to 0 transition (i.e. falling edge) is detected.
en	22	RW-V	0	Local Counter Enable.
ev_sel_ext	21	RW-V	0	Extentsion bit to the Event Select field.
ov_en	20	RW-V	0	When this bit is asserted and the corresponding counter overflows, its overflow bit is set in the local status register (PCU_PMON_BOX_STATUS.ov) and an overflow is sent on the message channel to the UBox. When the overflow is received by the UBox, the bit corresponding to this PCU will be set in U_MSR_PMON_GLOBAL_STATUS.ov_p.
rsv	19	RV	0	Reserved
edge_det	18	RW-V	0	When set to 1, rather than measuring the event in each cycle it is active, the corresponding counter will increment when a 0 to 1 transition (i.e. rising edge) is detected. When 0, the counter will increment in each cycle that the event is asserted. NOTE: .edge_det is in series following .thresh, Due to this, the .thresh field must be set to a non-0 value. For events that increment by no more than 1 per cycle, set .thresh to 0x1.
rst	17	WO	0	When set to 1, the corresponding counter will be cleared to 0.
rsv	16	RV	0	Reserved

Table 2-135. PCU_MSR_PMON_CTL{3-0} Register – Field Definitions (Sheet 2 of 2)

Field	Bits	Attr	HW Reset Val	Description
occ_sel	15:14	RW-V	0	Select which of three occupancy counters to use. 01 - Cores in C0 10 - Cores in C3 11 - Cores in C6
rsv	13:8	RV	0	Reserved
ev_sel	7:0	RW-V	0	Select event to be counted. NOTE: Bit 7 denotes whether the event requires the use of an occupancy subcounter.

The PCU performance monitor data registers are 48-bit wide. A counter overflow occurs when a carry out from bit 47 is detected. Software can force all uncore counting to freeze after N events by preloading a monitor with a count value of 2^{48} - N and setting the control register to send an overflow message to the UBox (refer to Section 2.1.1, "Counter Overflow"). During the interval of time between overflow and global disable, the counter value will wrap and continue to collect events.

If accessible, software can continuously read the data registers without disabling event collection.

Table 2-136. PCU_MSR_PMON_CTR{3-0} Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
rsv	63:48	RV	0	Reserved
event_count	47:0	RW-V	0	48-bit performance event counter

Context sensitive filtering is provided for through the PCU_MSR_PMON_BOX_FILTER register.

- For frequency/voltage band filters, the multiplier is at 100MHz granularity. So, a value of 32 (0x20) would represent a frequency of 3.2GHz.
- Support for limited Frequency/Voltage Band histogramming. Each of the four bands provided for in the filter may be simultaneous tracked by the corresponding event. Since use of the register as a filter is heavily overloaded, simultaneous application of this filter to additional events in the same run is severely limited

Table 2-137. PCU_MSR_PMON_BOX_FILTER Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
rsv	63:48	RV	0	Reserved
filt31_24	31:24	RW-V	0	Band 3 - For Voltage/Frequency Band Event
filt23_16	23:16	RW-V	0	Band 2 - For Voltage/Frequency Band Event
filt15_8	15:8	RW-V	0	Band 1 - For Voltage/Frequency Band Event
filt7_0	7:0	RW-V	0	Band 0 - For Voltage/Frequency Band Event

The PCU includes two extra MSRs that track the number of cycles a core (any core) is in either the C3 or C6 state. As mentioned before, these counters are not part of the PMON infrastructure so they can't be frozen or reset with the otherwise controlled by the PCU PMON control registers.

Note: To be clear, these counters track the number of cycles **some** core is in C3/6 state. It does not track the total number of cores in the C3/6 state in any cycle. For that, a user should refer to the regular PCU event list.

Table 2-138. PCU_MSR_CORE_C6_CTR Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
event_count	63:0	RW-V	0	64-bit performance event counter

Table 2-139. PCU_MSR_CORE_C3_CTR Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
event_count	63:0	RW-V	0	64-bit performance event counter

2.7.3 PCU Performance Monitoring Events

The PCU provides the ability to capture information covering a wide range of the PCU's functionality including:

- Number of cores in a given C-state per-cycle
- Core State Transitions - there are a larger number of events provided to track when cores transition C-state, when the enter/exit specific C-states, when they receive a C-state demotion, etc.
- Package State Transitions
- Frequency/Voltage Banding - ability to measure the number of cycles the uncore was operating within a frequency or voltage 'band' that can be specified in a separate filter register.

Note: Given the nature of many of the PCU events, a great deal of additional information can be measured by setting the *.edge_det* bit. By doing so, an event such as "Cycles Changing Frequency" becomes "Number of Frequency Transitions.

On Occupancy Events:

Because it is not possible to "sync" the PCU occupancy counters by employing tricks such as bus lock before the events start incrementing, the PCU has provided fixed occupancy counters to track the major queues.

1. Cores in C0 (4 bits)
2. Cores in C3 (4 bits)
3. Cores in C6 (4 bits)

The PCU perfmon implementation/programming is more complicated than many of the other units. As such, it is best to describe how to use them with a couple examples.

- Case 1: Cycles there was a Voltage Transition (Simple Event)
- Case 2: Cores in C0 (Occupancy Accumulation)
- Case 3: Cycles w/ more than 4 cores in C0 (Occupancy Thresholding)
- Case 4: Transitions into more than 4 cores in C0 (Thresholding + Edge Detect)

- Case 5: Cycles a) w/ > 4 Cores in C0 and b) there was a Voltage Transition
- Case 6: Cycles a) w/ <4 Cores in C0 and b) Freq < 2.0GHz

Table 2-140. PCU Configuration Examples

	Case					
Config	**1**	**2**	**3**	**4**	**5**	**6**
Counter Control 0						
.ev_sel		0x80	0x80	0x80	0x80	0x80
.occ_sel		0x1	0x1	0x1	0x1	0x1
.thresh		0x0	0x5	0x5	0x5	0x4
.invert		0	0	0	0	1
.occ_invert		0	0	0	0	1
.occ_edge_det		0	0	1	0	0
Counter Control 1						
.ev_sel	0x03				0x03	0x0B
Filter	0x00	0x00	0x00	0x00	0x00	0x14

2.7.4 PCU Box Events Ordered By Code

The following table summarizes the directly measured PCU Box events.

Symbol Name	Event Code	Ctrs	Extra Select Bit	Max Inc/Cyc	Description
CLOCKTICKS	0x00	0-3	0	1	pclk Cycles
FREQ_MAX_LIMIT_THERMAL_CYCLES	0x04	0-3	0	1	Thermal Strongest Upper Limit Cycles
FREQ_MAX_POWER_CYCLES	0x05	0-3	0	1	Power Strongest Upper Limit Cycles
FREQ_MAX_OS_CYCLES	0x06	0-3	0	1	OS Strongest Upper Limit Cycles
PROCHOT_INTERNAL_CYCLES	0x09	0-3	0	1	Internal Prochot
PROCHOT_EXTERNAL_CYCLES	0x0a	0-3	0	1	External Prochot
FREQ_BAND0_CYCLES	0x0b	0-3	0	1	Frequency Residency
FREQ_BAND1_CYCLES	0x0c	0-3	0	1	Frequency Residency
FREQ_BAND2_CYCLES	0x0d	0-3	0	1	Frequency Residency
FREQ_BAND3_CYCLES	0x0e	0-3	0	1	Frequency Residency
MEMORY_PHASE_SHEDDING_CYCLES	0x2f	0-3	0	1	Memory Phase Shedding Cycles
DEMOTIONS_CORE0	0x30	0-3	0	1	Core C State Demotions
DEMOTIONS_CORE1	0x31	0-3	0	1	Core C State Demotions
DEMOTIONS_CORE2	0x32	0-3	0	1	Core C State Demotions
DEMOTIONS_CORE3	0x33	0-3	0	1	Core C State Demotions
DEMOTIONS_CORE4	0x34	0-3	0	1	Core C State Demotions
DEMOTIONS_CORE5	0x35	0-3	0	1	Core C State Demotions
DEMOTIONS_CORE6	0x36	0-3	0	1	Core C State Demotions
DEMOTIONS_CORE7	0x37	0-3	0	1	Core C State Demotions

Symbol Name	Event Code	Ctrs	Extra Select Bit	Max Inc/Cyc	Description
DEMOTIONS_CORE8	0x38	0-3	0	1	Core C State Demotions
DEMOTIONS_CORE9	0x39	0-3	0	1	Core C State Demotions
DEMOTIONS_CORE10	0x3a	0-3	0	1	Core C State Demotions
DEMOTIONS_CORE11	0x3b	0-3	0	1	Core C State Demotions
DEMOTIONS_CORE12	0x3c	0-3	0	1	Core C State Demotions
DEMOTIONS_CORE13	0x3d	0-3	0	1	Core C State Demotions
DEMOTIONS_CORE14	0x3e	0-3	0	1	Core C State Demotions
DEMOTIONS_CORE15	0x3f	0-3	0	1	Core C State Demotions
DEMOTIONS_CORE16	0x40	0-3	0	1	Core C State Demotions
DEMOTIONS_CORE17	0x41	0-3	0	1	Core C State Demotions
VR_HOT_CYCLES	0x42	0-3	0	1	VR Hot
CORE0_TRANSITION_CYCLES	0x60	0-3	0	1	Core C State Transition Cycles
CORE1_TRANSITION_CYCLES	0x61	0-3	0	1	Core C State Transition Cycles
CORE2_TRANSITION_CYCLES	0x62	0-3	0	1	Core C State Transition Cycles
CORE3_TRANSITION_CYCLES	0x63	0-3	0	1	Core C State Transition Cycles
CORE4_TRANSITION_CYCLES	0x64	0-3	0	1	Core C State Transition Cycles
CORE5_TRANSITION_CYCLES	0x65	0-3	0	1	Core C State Transition Cycles
CORE6_TRANSITION_CYCLES	0x66	0-3	0	1	Core C State Transition Cycles
CORE7_TRANSITION_CYCLES	0x67	0-3	0	1	Core C State Transition Cycles
CORE8_TRANSITION_CYCLES	0x68	0-3	0	1	Core C State Transition Cycles
CORE9_TRANSITION_CYCLES	0x69	0-3	0	1	Core C State Transition Cycles
CORE10_TRANSITION_CYCLES	0x6a	0-3	0	1	Core C State Transition Cycles
CORE11_TRANSITION_CYCLES	0x6b	0-3	0	1	Core C State Transition Cycles
CORE12_TRANSITION_CYCLES	0x6c	0-3	0	1	Core C State Transition Cycles
CORE13_TRANSITION_CYCLES	0x6d	0-3	0	1	Core C State Transition Cycles
CORE14_TRANSITION_CYCLES	0x6e	0-3	0	1	Core C State Transition Cycles
CORE15_TRANSITION_CYCLES	0x6f	0-3	0	1	Core C State Transition Cycles
CORE16_TRANSITION_CYCLES	0x70	0-3	0	1	Core C State Transition Cycles
CORE17_TRANSITION_CYCLES	0x71	0-3	0	1	Core C State Transition Cycles
TOTAL_TRANSITION_CYCLES	0x72	0-3	0	1	Total Core C State Transition Cycles
FREQ_MIN_IO_P_CYCLES	0x73	0-3	0	1	IO P Limit Strongest Lower Limit Cycles
FREQ_TRANS_CYCLES	0x74	0-3	0	1	Cycles spent changing Frequency
FIVR_PS_PS0_CYCLES	0x75	0-3	0	1	Phase Shed 0 Cycles
FIVR_PS_PS1_CYCLES	0x76	0-3	0	1	Phase Shed 1 Cycles
FIVR_PS_PS2_CYCLES	0x77	0-3	0	1	Phase Shed 2 Cycles
FIVR_PS_PS3_CYCLES	0x78	0-3	0	1	Phase Shed 3 Cycles
UFS_TRANSITIONS_NO_CHANGE	0x79	0-3	0	1	
UFS_TRANSITIONS_UP_RING	0x7a	0-3	0	1	
UFS_TRANSITIONS_UP_STALL	0x7b	0-3	0	1	
UFS_TRANSITIONS_DOWN	0x7c	0-3	0	1	
UFS_TRANSITIONS_IO_P_LIMIT	0x7d	0-3	0	1	

Symbol Name	Event Code	Ctrs	Extra Select Bit	Max Inc/Cyc	Description
UFS_BANDWIDTH_MAX_RANGE	0x7e	0-3	0	1	
POWER_STATE_OCCUPANCY	0x80	0-3	0	8	Number of cores in C-State

2.7.5 PCU Box Common Metrics (Derived Events)

The following table summarizes metrics commonly calculated from PCU Box events.

Symbol Name: Definition	Equation
PCT_CYC_FREQ_CURRENT_LTD: Percentage of Cycles the Max Frequency is limited by current	FREQ_MAX_CURRENT_CYCLES / CLOCKTICKS
PCT_CYC_FREQ_OS_LTD: Percentage of Cycles the Max Frequency is limited by the OS	FREQ_MAX_OS_CYCLES / CLOCKTICKS
PCT_CYC_FREQ_POWER_LTD: Percentage of Cycles the Max Frequency is limited by power	FREQ_MAX_POWER_CYCLES / CLOCKTICKS
PCT_CYC_FREQ_THERMAL_LTD: Percentage of Cycles the Max Frequency is limited by thermal issues	FREQ_MAX_CURRENT_CYCLES / CLOCKTICKS

2.7.6 PCU Box Performance Monitor Event List

The section enumerates Intel® Xeon® Processor D-1500 Product Family performance monitoring events for the PCU Box.

CLOCKTICKS

- **Title:** pclk Cycles
- **Category:** PCLK Events
- **Event Code:** 0x00
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** The PCU runs off a fixed 800 MHz clock. This event counts the number of pclk cycles measured while the counter was enabled. The pclk, like the Memory Controller's dclk, counts at a constant rate making it a good measure of actual wall time.

CORE0_TRANSITION_CYCLES

- **Title:** Core C State Transition Cycles
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x60
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of cycles spent performing core C state transitions. There is one event per core.
- **NOTE:** This only tracks the hardware portion in the RCFSM (CFCFSM). This portion is just doing the core C state transition. It does not include any necessary frequency/voltage transitions.

CORE10_TRANSITION_CYCLES

- **Title:** Core C State Transition Cycles
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x6a
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3

- **Definition:** Number of cycles spent performing core C state transitions. There is one event per core.

CORE11_TRANSITION_CYCLES

- **Title:** Core C State Transition Cycles
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x6b
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of cycles spent performing core C state transitions. There is one event per core.

CORE12_TRANSITION_CYCLES

- **Title:** Core C State Transition Cycles
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x6c
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of cycles spent performing core C state transitions. There is one event per core.

CORE13_TRANSITION_CYCLES

- **Title:** Core C State Transition Cycles
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x6d
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of cycles spent performing core C state transitions. There is one event per core.

CORE14_TRANSITION_CYCLES

- **Title:** Core C State Transition Cycles
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x6e
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of cycles spent performing core C state transitions. There is one event per core.

CORE15_TRANSITION_CYCLES

- **Title:** Core C State Transition Cycles
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x6f
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of cycles spent performing core C state transitions. There is one event per core.

CORE16_TRANSITION_CYCLES

- **Title:** Core C State Transition Cycles
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x70
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of cycles spent performing core C state transitions. There is one event per core.

CORE17_TRANSITION_CYCLES

- **Title:** Core C State Transition Cycles
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x71
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of cycles spent performing core C state transitions. There is one event per core.

CORE1_TRANSITION_CYCLES

- **Title:** Core C State Transition Cycles
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x61
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of cycles spent performing core C state transitions. There is one event per core.

CORE2_TRANSITION_CYCLES

- **Title:** Core C State Transition Cycles
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x62
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of cycles spent performing core C state transitions. There is one event per core.

CORE3_TRANSITION_CYCLES

- **Title:** Core C State Transition Cycles
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x63
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of cycles spent performing core C state transitions. There is one event per core.

CORE4_TRANSITION_CYCLES

- **Title:** Core C State Transition Cycles
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x64
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of cycles spent performing core C state transitions. There is one event per core.

CORE5_TRANSITION_CYCLES

- **Title:** Core C State Transition Cycles
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x65
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of cycles spent performing core C state transitions. There is one event per core.

CORE6_TRANSITION_CYCLES

- **Title:** Core C State Transition Cycles
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x66
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of cycles spent performing core C state transitions. There is one event per core.

CORE7_TRANSITION_CYCLES

- **Title:** Core C State Transition Cycles
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x67
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of cycles spent performing core C state transitions. There is one event per core.

CORE8_TRANSITION_CYCLES

- **Title:** Core C State Transition Cycles
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x68
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of cycles spent performing core C state transitions. There is one event per core.
- **NOTE:** This only tracks the hardware portion in the RCFSM (CFCFSM). This portion is just doing the core C state transition. It does not include any necessary frequency/voltage transitions.

CORE9_TRANSITION_CYCLES

- **Title:** Core C State Transition Cycles
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x69
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of cycles spent performing core C state transitions. There is one event per core.

DEMOTIONS_CORE0

- **Title:** Core C State Demotions
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x30
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when a configurable cores had a C-state demotion

DEMOTIONS_CORE1

- **Title:** Core C State Demotions
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x31
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when a configurable cores had a C-state demotion

DEMOTIONS_CORE10

- **Title:** Core C State Demotions
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x3a
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when a configurable cores had a C-state demotion

DEMOTIONS_CORE11

- **Title:** Core C State Demotions
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x3b
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when a configurable cores had a C-state demotion

DEMOTIONS_CORE12

- **Title:** Core C State Demotions
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x3c
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when a configurable cores had a C-state demotion

DEMOTIONS_CORE13

- **Title:** Core C State Demotions
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x3d
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when a configurable cores had a C-state demotion

DEMOTIONS_CORE14

- **Title:** Core C State Demotions
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x3e
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when a configurable cores had a C-state demotion

DEMOTIONS_CORE15

- **Title:** Core C State Demotions
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x3f
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when a configurable cores had a C-state demotion

DEMOTIONS_CORE16

- **Title:** Core C State Demotions
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x40
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when a configurable cores had a C-state demotion

DEMOTIONS_CORE17

- **Title:** Core C State Demotions
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x41
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when a configurable cores had a C-state demotion

DEMOTIONS_CORE2

- **Title:** Core C State Demotions
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x32
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when a configurable cores had a C-state demotion

DEMOTIONS_CORE3

- **Title:** Core C State Demotions
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x33
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when a configurable cores had a C-state demotion

DEMOTIONS_CORE4

- **Title:** Core C State Demotions
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x34
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when a configurable cores had a C-state demotion

DEMOTIONS_CORE5

- **Title:** Core C State Demotions
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x35
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when a configurable cores had a C-state demotion

DEMOTIONS_CORE6

- **Title:** Core C State Demotions
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x36
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when a configurable cores had a C-state demotion

DEMOTIONS_CORE7

- **Title:** Core C State Demotions
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x37
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when a configurable cores had a C-state demotion

DEMOTIONS_CORE8

- **Title:** Core C State Demotions
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x38
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when a configurable cores had a C-state demotion

DEMOTIONS_CORE9

- **Title:** Core C State Demotions
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x39
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when a configurable cores had a C-state demotion

FIVR_PS_PS0_CYCLES

- **Title:** Phase Shed 0 Cycles
- **Category:** FIVR Events
- **Event Code:** 0x75
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Cycles spent in phase-shedding power state 0

FIVR_PS_PS1_CYCLES

- **Title:** Phase Shed 1 Cycles
- **Category:** FIVR Events
- **Event Code:** 0x76
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Cycles spent in phase-shedding power state 1

FIVR_PS_PS2_CYCLES

- **Title:** Phase Shed 2 Cycles
- **Category:** FIVR Events
- **Event Code:** 0x77
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Cycles spent in phase-shedding power state 2

FIVR_PS_PS3_CYCLES

- **Title:** Phase Shed 3 Cycles
- **Category:** FIVR Events
- **Event Code:** 0x78
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Cycles spent in phase-shedding power state 3

FREQ_BAND0_CYCLES

- **Title:** Frequency Residency
- **Category:** FREQ_RESIDENCY Events
- **Event Code:** 0x0b
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Filter Dependency:** PCUFilter[7:0]

- **Definition:** Counts the number of cycles that the uncore was running at a frequency greater than or equal to the frequency that is configured in the filter. One can use all four counters with this event, so it is possible to track up to 4 configurable bands. One can use edge detect in conjunction with this event to track the number of times that we transitioned into a frequency greater than or equal to the configurable frequency. One can also use inversion to track cycles when we were less than the configured frequency.
- **NOTE:** The PMON control registers in the PCU only update on a frequency transition. Changing the measuring threshold during a sample interval may introduce errors in the counts. This is especially true when running at a constant frequency for an extended period of time. There is a corner case here: we set this code on the GV transition. So, if we never GV we will never call this code. This event does not include transition times. It is handled on fast path.

FREQ_BAND1_CYCLES

- **Title:** Frequency Residency
- **Category:** FREQ_RESIDENCY Events
- **Event Code:** 0x0c
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Filter Dependency:** PCUFilter[15:8]
- **Definition:** Counts the number of cycles that the uncore was running at a frequency greater than or equal to the frequency that is configured in the filter. One can use all four counters with this event, so it is possible to track up to 4 configurable bands. One can use edge detect in conjunction with this event to track the number of times that we transitioned into a frequency greater than or equal to the configurable frequency. One can also use inversion to track cycles when we were less than the configured frequency.
- **NOTE:** The PMON control registers in the PCU only update on a frequency transition. Changing the measuring threshold during a sample interval may introduce errors in the counts. This is especially true when running at a constant frequency for an extended period of time. There is a corner case here: we set this code on the GV transition. So, if we never GV we will never call this code. This event does not include transition times. It is handled on fast path.

FREQ_BAND2_CYCLES

- **Title:** Frequency Residency
- **Category:** FREQ_RESIDENCY Events
- **Event Code:** 0x0d
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Filter Dependency:** PCUFilter[23:16]
- **Definition:** Counts the number of cycles that the uncore was running at a frequency greater than or equal to the frequency that is configured in the filter. One can use all four counters with this event, so it is possible to track up to 4 configurable bands. One can use edge detect in conjunction with this event to track the number of times that we transitioned into a frequency greater than or equal to the configurable frequency. One can also use inversion to track cycles when we were less than the configured frequency.
- **NOTE:** The PMON control registers in the PCU only update on a frequency transition. Changing the measuring threshold during a sample interval may introduce errors in the counts. This is especially true when running at a constant frequency for an extended period of time. There is a corner case here: we set this code on the GV transition. So, if we never GV we will never call this code. This event does not include transition times. It is handled on fast path.

FREQ_BAND3_CYCLES

- **Title:** Frequency Residency
- **Category:** FREQ_RESIDENCY Events
- **Event Code:** 0x0e
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Filter Dependency:** PCUFilter[31:24]
- **Definition:** Counts the number of cycles that the uncore was running at a frequency greater than or equal to the frequency that is configured in the filter. One can use all four counters with this event, so it is possible to track up to 4 configurable bands. One can use edge detect in conjunction

with this event to track the number of times that we transitioned into a frequency greater than or equal to the configurable frequency. One can also use inversion to track cycles when we were less than the configured frequency.

- **NOTE:** The PMON control registers in the PCU only update on a frequency transition. Changing the measuring threshold during a sample interval may introduce errors in the counts. This is especially true when running at a constant frequency for an extended period of time. There is a corner case here: we set this code on the GV transition. So, if we never GV we will never call this code. This event does not include transition times. It is handled on fast path.

FREQ_MAX_LIMIT_THERMAL_CYCLES

- **Title:** Thermal Strongest Upper Limit Cycles
- **Category:** FREQ_MAX_LIMIT Events
- **Event Code:** 0x04
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles when thermal conditions are the upper limit on frequency. This is related to the THERMAL_THROTTLE CYCLES_ABOVE_TEMP event, which always counts cycles when we are above the thermal temperature. This event (STRONGEST_UPPER_LIMIT) is sampled at the output of the algorithm that determines the actual frequency, while THERMAL_THROTTLE looks at the input.

FREQ_MAX_OS_CYCLES

- **Title:** OS Strongest Upper Limit Cycles
- **Category:** FREQ_MAX_LIMIT Events
- **Event Code:** 0x06
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles when the OS is the upper limit on frequency.
- **NOTE:** Essentially, this event says the OS is getting the frequency it requested.

FREQ_MAX_POWER_CYCLES

- **Title:** Power Strongest Upper Limit Cycles
- **Category:** FREQ_MAX_LIMIT Events
- **Event Code:** 0x05
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles when power is the upper limit on frequency.

FREQ_MIN_IO_P_CYCLES

- **Title:** IO P Limit Strongest Lower Limit Cycles
- **Category:** FREQ_MIN_LIMIT Events
- **Event Code:** 0x73
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles when IO P Limit is preventing us from dropping the frequency lower. This algorithm monitors the needs to the IO subsystem on both local and remote sockets and will maintain a frequency high enough to maintain good IO BW. This is necessary for when all the IA cores on a socket are idle but a user still would like to maintain high IO Bandwidth.

FREQ_TRANS_CYCLES

- **Title:** Cycles spent changing Frequency
- **Category:** FREQ_TRANS Events
- **Event Code:** 0x74
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles when the system is changing frequency. This can not be filtered by thread ID. One can also use it with the occupancy counter that monitors number of threads in C0 to estimate the performance impact that frequency transitions had on the system.

MEMORY_PHASE_SHEDDING_CYCLES

- **Title:** Memory Phase Shedding Cycles
- **Category:** MEMORY_PHASE_SHEDDING Events
- **Event Code:** 0x2f
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles that the PCU has triggered memory phase shedding. This is a mode that can be run in the iMC physicals that saves power at the expense of additional latency.
- **NOTE:** Package C1

POWER_STATE_OCCUPANCY

- **Title:** Number of cores in C-State
- **Category:** POWER_STATE_OCC Events
- **Event Code:** 0x80
- **Max. Inc/Cyc:.** 8, **Register Restrictions:** 0-3
- **Definition:** This is an occupancy event that tracks the number of cores that are in the chosen C-State. It can be used by itself to get the average number of cores in that C-state with threshholding to generate histograms, or with other PCU events and occupancy triggering to capture other details.

Table 2-141. Unit Masks for POWER_STATE_OCCUPANCY

Extension	umask [15:8]	Description
CORES_C0	b01000000	C0 and C1
CORES_C3	b10000000	C3
CORES_C6	b11000000	C6 and C7

PROCHOT_EXTERNAL_CYCLES

- **Title:** External Prochot
- **Category:** PROCHOT Events
- **Event Code:** 0x0a
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles that we are in external PROCHOT mode. This mode is triggered when a sensor off the die determines that something off-die (like DRAM) is too hot and must throttle to avoid damaging the chip.

PROCHOT_INTERNAL_CYCLES

- **Title:** Internal Prochot
- **Category:** PROCHOT Events
- **Event Code:** 0x09
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles that we are in Internal PROCHOT mode. This mode is triggered when a sensor on the die determines that we are too hot and must throttle to avoid damaging the chip.

TOTAL_TRANSITION_CYCLES

- **Title:** Total Core C State Transition Cycles
- **Category:** CORE_C_STATE_TRANSITION Events
- **Event Code:** 0x72
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Number of cycles spent performing core C state transitions across all cores.

UFS_BANDWIDTH_MAX_RANGE

- **Title:**
- **Category:** UFS Events
- **Event Code:** 0x7e
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

UFS_TRANSITIONS_DOWN

- **Title:**
- **Category:** UFS Events
- **Event Code:** 0x7c
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Ring GV down (due to low ring traffic)

UFS_TRANSITIONS_IO_P_LIMIT

- **Title:**
- **Category:** UFS Events
- **Event Code:** 0x7d
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

UFS_TRANSITIONS_NO_CHANGE

- **Title:**
- **Category:** UFS Events
- **Event Code:** 0x79
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Ring GV with same final and initial frequency

UFS_TRANSITIONS_UP_RING

- **Title:**
- **Category:** UFS Events
- **Event Code:** 0x7a
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Ring GV up due to high ring traffic

UFS_TRANSITIONS_UP_STALL

- **Title:**
- **Category:** UFS Events
- **Event Code:** 0x7b
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Ring GV up due to high core stalls

VR_HOT_CYCLES

- **Title:** VR Hot
- **Category:** VR_HOT Events
- **Event Code:** 0x42
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:**

2.8 R2PCIe Performance Monitoring

R2PCIe represents the interface between the Ring and IIO traffic to/from PCIe.

2.8.1 R2PCIe Performance Monitoring Overview

The R2PCIe Box supports event monitoring through four 48b wide counters (R2_PCI_PMON_CTR/CTL{3:0}). Each of these four counters can be programmed to count almost any R2PCIe event (see NOTE for exceptions). the R2PCIe counters can increment by a maximum of 5b per cycle.

For information on how to setup a monitoring session, refer to Section 2.1, "Uncore Per-Socket Performance Monitoring Control".

Note: Only counter 0 can be used for tracking occupancy events. Only counters 2&3 can be used for ring utilization events.

2.8.1.1 R2PCIe PMON Registers - On Overflow and the Consequences (PMI/Freeze)

If an overflow is detected from a R2PCIe performance counter, the overflow bit is set at the box level (R2_PCI_PMON_CTL.ov). If the counter is enabled to communicate the overflow (R2_PCI_PMON_CTL.ov_en is set to 1), an overflow message is sent to the UBox. When the UBox receives the overflow signal, the U_MSR_PMON_GLOBAL_STATUS.ov_rp bit is set (see Table 2-3, "U_MSR_PMON_GLOBAL_STATUS Register – Field Definitions"), a global freeze signal is sent and a PMI can be generated.

Once a freeze has occurred, in order to see a new freeze, the overflow responsible for the freeze must be cleared by setting the corresponding bit in R2_PCI_PMON_BOX_STATUS.ov and U_MSR_PMON_GLOBAL_STATUS.ov_rp to 1 (which acts to clear the bits). Assuming all the counters have been locally enabled (*.en* bit in control registers meant to monitor events) and the overflow bits have been cleared, the R2PCIe is prepared for a new sample interval. Once the global controls have been re-enabled (Section 2.1.4, "Enabling a New Sample Interval from Frozen Counters"), counting will resume.

2.8.2 R2PCIe Performance Monitors

Table 2-142. R2PCIe Performance Monitoring Registers (PCICFG) (Sheet 1 of 2)

Register Name	PCICFG Address	Size (bits)	Description
PCICFG Base Address	Dev:Func DeviceID		
R2PCIe PMON Registers	D16:F1 0x6F34		
Box-Level Control/Status			
R2_PCI_PMON_BOX_STATUS	F8	32	R2PCIe PMON Box-Wide Status
R2_PCI_PMON_BOX_CTL	F4	32	R2PCIe PMON Box-Wide Control
Generic Counter Control			
R2_PCI_PMON_CTL3	E4	32	R2PCIe PMON Control for Counter 3

Table 2-142. R2PCIe Performance Monitoring Registers (PCICFG) (Sheet 2 of 2)

Register Name	PCICFG Address	Size (bits)	Description
R2_PCI_PMON_CTL2	E0	32	R2PCIe PMON Control for Counter 2
R2_PCI_PMON_CTL1	DC	32	R2PCIe PMON Control for Counter 1
R2_PCI_PMON_CTL0	D8	32	R2PCIe PMON Control for Counter 0
Generic Counters			
R2_PCI_PMON_CTR3	BC+B8	32x2	R2PCIe PMON Counter 3
R2_PCI_PMON_CTR2	B4+B0	32x2	R2PCIe PMON Counter 2
R2_PCI_PMON_CTR1	AC+A8	32x2	R2PCIe PMON Counter 1
R2_PCI_PMON_CTR0	A4+A0	32x2	R2PCIe PMON Counter 0

2.8.2.1 R2PCIe Box Level PMON State

The following registers represent the state governing all box-level PMUs in the R2PCIe Box.

In the case of the R2PCIe, the R2_PCI_PMON_BOX_CTL register provides the ability to manually freeze the counters in the box (*.frz*) and reset the generic state (*.rst_ctrs* and *.rst_ctrl*).

If an overflow is detected from one of the R2PCIe PMON registers, the corresponding bit in the R2_PCI_PMON_BOX_STATUS.ov field will be set. To reset these overflow bits, a user must write a value of '1' to them (which will clear the bits).

Table 2-143. R2_PCI_PMON_BOX_CTL Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
ig	31:9	RV	0	Reserved
frz	8	WO	0	Freeze. If set to 1 the counters in this box will be frozen.
ig	7:2	RV	0	Reserved
rst_ctrs	1	WO	0	Reset Counters. When set to 1, the Counter Registers will be reset to 0.
rst_ctrl	0	WO	0	Reset Control. When set to 1, the Counter Control Registers will be reset to 0.

Table 2-144. R2_PCI_PMON_BOX_STATUS Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
ig	31:4	RV	0	Reserved
ov	3:0	RW1C	0	If an overflow is detected from the corresponding R2_PCI_PMON_CTR register, it's overflow bit will be set. NOTE: Write of '1' will clear the bit.

2.8.2.2 R2PCIe PMON state - Counter/Control Pairs

The following table defines the layout of the R2PCIe performance monitor control registers. The main task of these configuration registers is to select the event to be monitored by their respective data counter (*.ev_sel*, *.umask*). Additional control bits are provided to shape the incoming events (e.g. *.invert*, *.edge_det*, *.thresh*) as well as provide additional functionality for monitoring software (*.rst*,*.ov_en*).

Table 2-145. R2_PCI_PMON_CTL{3-0} Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
thresh	31:24	RW-V	0	Threshold used in counter comparison.
invert	23	RW-V	0	Invert comparison against Threshold. 0 - comparison will be 'is event increment >= threshold?'. 1 - comparison is inverted - 'is event increment < threshold?' NOTE: .invert is in series following .thresh, Due to this, the .thresh field must be set to a non-0 value. For events that increment by no more than 1 per cycle, set .thresh to 0x1. Also, if .edge_det is set to 1, the counter will increment when a 1 to 0 transition (i.e. falling edge) is detected.
en	22	RW-V	0	Local Counter Enable.
ig	21	RV	0	Reserved. SW must write to 0 else behavior is undefined.
ov_en	20	RW-V	0	When this bit is asserted and the corresponding counter overflows, its overflow bit is set in the local status register (R2_PCI_PMON_BOX_STATUS.ov) and an overflow is sent on the message channel to the UBox. When the overflow is received by the UBox, the bit corresponding to this R2 will be set in U_MSR_PMON_GLOBAL_STATUS.ov_rp
ig	19	RV	0	Reserved
edge_det	18	RW-V	0	When set to 1, rather than measuring the event in each cycle it is active, the corresponding counter will increment when a 0 to 1 transition (i.e. rising edge) is detected. When 0, the counter will increment in each cycle that the event is asserted. NOTE: .edge_det is in series following .thresh, Due to this, the .thresh field must be set to a non-0 value. For events that increment by no more than 1 per cycle, set .thresh to 0x1.
rst	17	WO	0	When set to 1, the corresponding counter will be cleared to 0.
rsv	16	RV	0	Reserved. SW must write to 0 else behavior is undefined.
umask	15:8	RW-V	0	Select subevents to be counted within the selected event.
ev_sel	7:0	RW-V	0	Select event to be counted.

Note: Due to a issue found with the Intel® Xeon® Processor D-1500 Product Family hardware, it will be necessary to write each control register twice in a row in order for the Event Select field to take hold. It is recommended that SW perform the first write with the enable bit set to 0 followed by a write of the same control register value but with the enable bit set to 1.

The R2PCIe performance monitor data registers are 48-bit wide. A counter overflow occurs when a carry out from bit 47 is detected. Software can force all uncore counting to freeze after N events by preloading a monitor with a count value of 2^{48} - N and setting the control register to send an overflow message to the UBox (Section 2.1.1.1, "Freezing on Counter Overflow"). During the interval of time between overflow and global disable, the counter value will wrap and continue to collect events.

If accessible, software can continuously read the data registers without disabling event collection.

Table 2-146. R2_PCI_PMON_CTR{3-0} Register – Field Definitions

Field	Bits	Attr	HW Reset Val	Description
ig	63:48	RV	0	Ignored
event_count	47:0	RW-V	0	48-bit performance event counter

2.8.3 R2PCIe Performance Monitoring Events

R2PCIe provides events to track information related to all the traffic passing through it's boundaries.

- IIO credit tracking - credits rejected, acquired and used all broken down by message Class.
- Ring Stop Events
  To track Ingress/Egress Traffic and Ring Utilization (broken down by direction and ring type) statistics.

2.8.4 R2PCIe Box Events Ordered By Code

The following table summarizes the directly measured R2PCIe Box events.

Symbol Name	Event Code	Ctrs	Max Inc/Cyc	Description
CLOCKTICKS	0x01	0-3	1	Number of uclks in domain
RING_AD_USED	0x07	0-3	1	R2 AD Ring in Use
RING_AK_USED	0x08	0-3	1	R2 AK Ring in Use
RING_BL_USED	0x09	0-3	1	R2 BL Ring in Use
RING_IV_USED	0x0a	0-3	1	R2 IV Ring in Use
RxR_CYCLES_NE	0x10	0-1	1	Ingress Cycles Not Empty
RxR_INSERTS	0x11	0-1	1	Ingress Allocations
RING_AK_BOUNCES	0x12	0-3	1	AK Ingress Bounced
RxR_OCCUPANCY	0x13	0	24	Ingress Occupancy Accumulator
TxR_CYCLES_NE	0x23	0	1	Egress Cycles Not Empty
TxR_CYCLES_FULL	0x25	0	1	Egress Cycles Full
TxR_NACK_CW	0x26	0-1	1	Egress CCW NACK
IIO_CREDIT	0x2d	0-1	4	

2.8.5 R2PCIe Box Common Metrics (Derived Events)

The following table summarizes metrics commonly calculated from R2PCIe Box events.

Symbol Name: Definition	Equation
CYC_USED_DN: Cycles Used in the Down direction, Even polarity	RING_BL_USED.CCW / SAMPLE_INTERVAL
CYC_USED_UP: Cycles Used in the Up direction, Even polarity	RING_BL_USED.CW / SAMPLE_INTERVAL

Symbol Name: Definition	Equation
RING_THRU_DN_BYTES: Ring throughput in the Down direction, Even polarity in Bytes	RING_BL_USED.CCW* 32
RING_THRU_UP_BYTES: Ring throughput in the Up direction, Even polarity in Bytes	RING_BL_USED.CW * 32

2.8.6 R2PCIe Box Performance Monitor Event List

The section enumerates Intel® Xeon® Processor D-1500 Product Family performance monitoring events for the R2PCIe Box.

CLOCKTICKS

- **Title:** Number of uclks in domain
- **Category:** UCLK Events
- **Event Code:** 0x01
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of uclks in the R2PCIe uclk domain. This could be slightly different than the count in the Ubox because of enable/freeze delays. However, because the R2PCIe is close to the Ubox, they generally should not diverge by more than a handful of cycles.

IIO_CREDIT

- **Title:**
- **Category:** IIO Credit Events
- **Event Code:** 0x2d
- **Max. Inc/Cyc:.** 4, **Register Restrictions:** 0-1
- **Definition:**

Table 2-147. Unit Masks for IIO_CREDIT

Extension	umask [15:8]	Description
PRQ_QPI0	bxxxxxxx1	
PRQ_QPI1	bxxxxxx1x	
ISOCH_QPI0	bxxxxx1xx	
ISOCH_QPI1	bxxxx1xxx	

RING_AD_USED

- **Title:** R2 AD Ring in Use
- **Category:** RING Events
- **Event Code:** 0x07
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles that the AD ring is being used at this ring stop. This includes when packets are passing by and when packets are being sunk, but does not include when packets are being sent from the ring stop.
- **NOTE:** In any cycle, a ring stop can see up to one packet moving in the CW direction and one packet moving in the CCW direction.

Table 2-148. Unit Masks for RING_AD_USED

Extension	umask [15:8]	Description
CW_EVEN	bxxxxxxx1	Clockwise and Even Filters for the Clockwise and Even ring polarity.
CW_ODD	bxxxxxx1x	Clockwise and Odd Filters for the Clockwise and Odd ring polarity.
CW	b00000011	Clockwise
CCW_EVEN	bxxxxx1xx	Counterclockwise and Even Filters for the Counterclockwise and Even ring polarity.
CCW_ODD	bxxxx1xxx	Counterclockwise and Odd Filters for the Counterclockwise and Odd ring polarity.
CCW	b00001100	Counterclockwise

RING_AK_BOUNCES

- **Title:** AK Ingress Bounced
- **Category:** RING Events
- **Event Code:** 0x12
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of times when a request destined for the AK ingress bounced.

Table 2-149. Unit Masks for RING_AK_BOUNCES

Extension	umask [15:8]	Description
UP	bxxxxxxx1	Up
DN	bxxxxxx1x	Dn

RING_AK_USED

- **Title:** R2 AK Ring in Use
- **Category:** RING Events
- **Event Code:** 0x08
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles that the AK ring is being used at this ring stop. This includes when packets are passing by and when packets are being sunk, but does not include when packets are being sent from the ring stop.
- **NOTE:** In any cycle, a ring stop can see up to one packet moving in the CW direction and one packet moving in the CCW direction.

Table 2-150. Unit Masks for RING_AK_USED

Extension	umask [15:8]	Description
CW_EVEN	bxxxxxxx1	Clockwise and Even Filters for the Clockwise and Even ring polarity.
CW_ODD	bxxxxxx1x	Clockwise and Odd Filters for the Clockwise and Odd ring polarity.
CW	b00000011	Clockwise
CCW_EVEN	bxxxxx1xx	Counterclockwise and Even Filters for the Counterclockwise and Even ring polarity.

Table 2-150. Unit Masks for RING_AK_USED

Extension	umask [15:8]	Description
CCW_ODD	bxxxx1xxx	Counterclockwise and Odd Filters for the Counterclockwise and Odd ring polarity.
CCW	b00001100	Counterclockwise

RING_BL_USED

- **Title:** R2 BL Ring in Use
- **Category:** RING Events
- **Event Code:** 0x09
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles that the BL ring is being used at this ring stop. This includes when packets are passing by and when packets are being sunk, but does not include when packets are being sent from the ring stop.
- **NOTE:** In any cycle, a ring stop can see up to one packet moving in the CW direction and one packet moving in the CCW direction.

Table 2-151. Unit Masks for RING_BL_USED

Extension	umask [15:8]	Description
CW_EVEN	bxxxxxxx1	Clockwise and Even Filters for the Clockwise and Even ring polarity.
CW_ODD	bxxxxxx1x	Clockwise and Odd Filters for the Clockwise and Odd ring polarity.
CW	b00000011	Clockwise
CCW_EVEN	bxxxxx1xx	Counterclockwise and Even Filters for the Counterclockwise and Even ring polarity.
CCW_ODD	bxxxx1xxx	Counterclockwise and Odd Filters for the Counterclockwise and Odd ring polarity.
CCW	b00001100	Counterclockwise

RING_IV_USED

- **Title:** R2 IV Ring in Use
- **Category:** RING Events
- **Event Code:** 0x0a
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-3
- **Definition:** Counts the number of cycles that the IV ring is being used at this ring stop. This includes when packets are passing by and when packets are being sent, but does not include when packets are being sunk into the ring stop.
- **NOTE:** IV messages are split into two parts. In any cycle, a ring stop can see up to one (half-)packet moving in the CW direction and one (half-)packet moving in the CCW direction.

Table 2-152. Unit Masks for RING_IV_USED

Extension	umask [15:8]	Description
CW	b00000011	Clockwise
CCW	b00001100	Counterclockwise
ANY	b00001111	Any

RxR_CYCLES_NE

- **Title:** Ingress Cycles Not Empty
- **Category:** INGRESS Events
- **Event Code:** 0x10
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** Counts the number of cycles when the R2PCIe Ingress is not empty. This tracks one of the three rings that are used by the R2PCIe agent. This can be used in conjunction with the R2PCIe Ingress Occupancy Accumulator event in order to calculate average queue occupancy. Multiple ingress buffers can be tracked at a given time using multiple counters.

Table 2-153. Unit Masks for RxR_CYCLES_NE

Extension	umask [15:8]	Description
NCB	bxxx1xxxx	NCB NCB Ingress Queue
NCS	bxx1xxxxx	NCS NCS Ingress Queue

RxR_INSERTS

- **Title:** Ingress Allocations
- **Category:** INGRESS Events
- **Event Code:** 0x11
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:** Counts the number of allocations into the R2PCIe Ingress. This tracks one of the three rings that are used by the R2PCIe agent. This can be used in conjunction with the R2PCIe Ingress Occupancy Accumulator event in order to calculate average queue latency. Multiple ingress buffers can be tracked at a given time using multiple counters.

Table 2-154. Unit Masks for RxR_INSERTS

Extension	umask [15:8]	Description
NCB	bxxx1xxxx	NCB NCB Ingress Queue
NCS	bxx1xxxxx	NCS NCS Ingress Queue

RxR_OCCUPANCY

- **Title:** Ingress Occupancy Accumulator
- **Category:** INGRESS Events
- **Event Code:** 0x13
- **Max. Inc/Cyc:.** 24, **Register Restrictions:** 0
- **Definition:** Accumulates the occupancy of a given R2PCIe Ingress queue in each cycles. This tracks one of the three ring Ingress buffers. This can be used with the R2PCIe Ingress Not Empty event to calculate average occupancy or the R2PCIe Ingress Allocations event in order to calculate average queuing latency.

Table 2-155. Unit Masks for RxR_OCCUPANCY

Extension	umask [15:8]	Description
DRS	b00001000	DRS DRS Ingress Queue

TxR_CYCLES_FULL

- **Title:** Egress Cycles Full
- **Category:** EGRESS Events
- **Event Code:** 0x25
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0
- **Definition:** Counts the number of cycles when the R2PCIe Egress buffer is full.

Table 2-156. Unit Masks for TxR_CYCLES_FULL

Extension	umask [15:8]	Description
AD	bxxxxxxx1	AD AD Egress Queue
AK	bxxxxxx1x	AK AK Egress Queue
BL	bxxxxx1xx	BL BL Egress Queue

TxR_CYCLES_NE

- **Title:** Egress Cycles Not Empty
- **Category:** EGRESS Events
- **Event Code:** 0x23
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0
- **Definition:** Counts the number of cycles when the R2PCIe Egress is not empty. This tracks one of the three rings that are used by the R2PCIe agent. This can be used in conjunction with the R2PCIe Egress Occupancy Accumulator event in order to calculate average queue occupancy. Only a single Egress queue can be tracked at any given time. It is not possible to filter based on direction or polarity.

Table 2-157. Unit Masks for TxR_CYCLES_NE

Extension	umask [15:8]	Description
AD	bxxxxxxx1	AD AD Egress Queue
AK	bxxxxxx1x	AK AK Egress Queue
BL	bxxxxx1xx	BL BL Egress Queue

TxR_NACK_CW

- **Title:** Egress CCW NACK
- **Category:** EGRESS Events
- **Event Code:** 0x26
- **Max. Inc/Cyc:.** 1, **Register Restrictions:** 0-1
- **Definition:**

Table 2-158. Unit Masks for TxR_NACK_CW

Extension	umask [15:8]	Description
DN_AD	bxxxxxxx1	AD CCW AD CounterClockwise Egress Queue
DN_BL	bxxxxxx1x	BL CCW BL CounterClockwise Egress Queue
DN_AK	bxxxxx1xx	AK CCW AK CounterClockwise Egress Queue
UP_AD	bxxxx1xxx	AK CCW BL CounterClockwise Egress Queue
UP_BL	bxxx1xxxx	BL CCW AD CounterClockwise Egress Queue
UP_AK	bxx1xxxxx	BL CW AD Clockwise Egress Queue

§