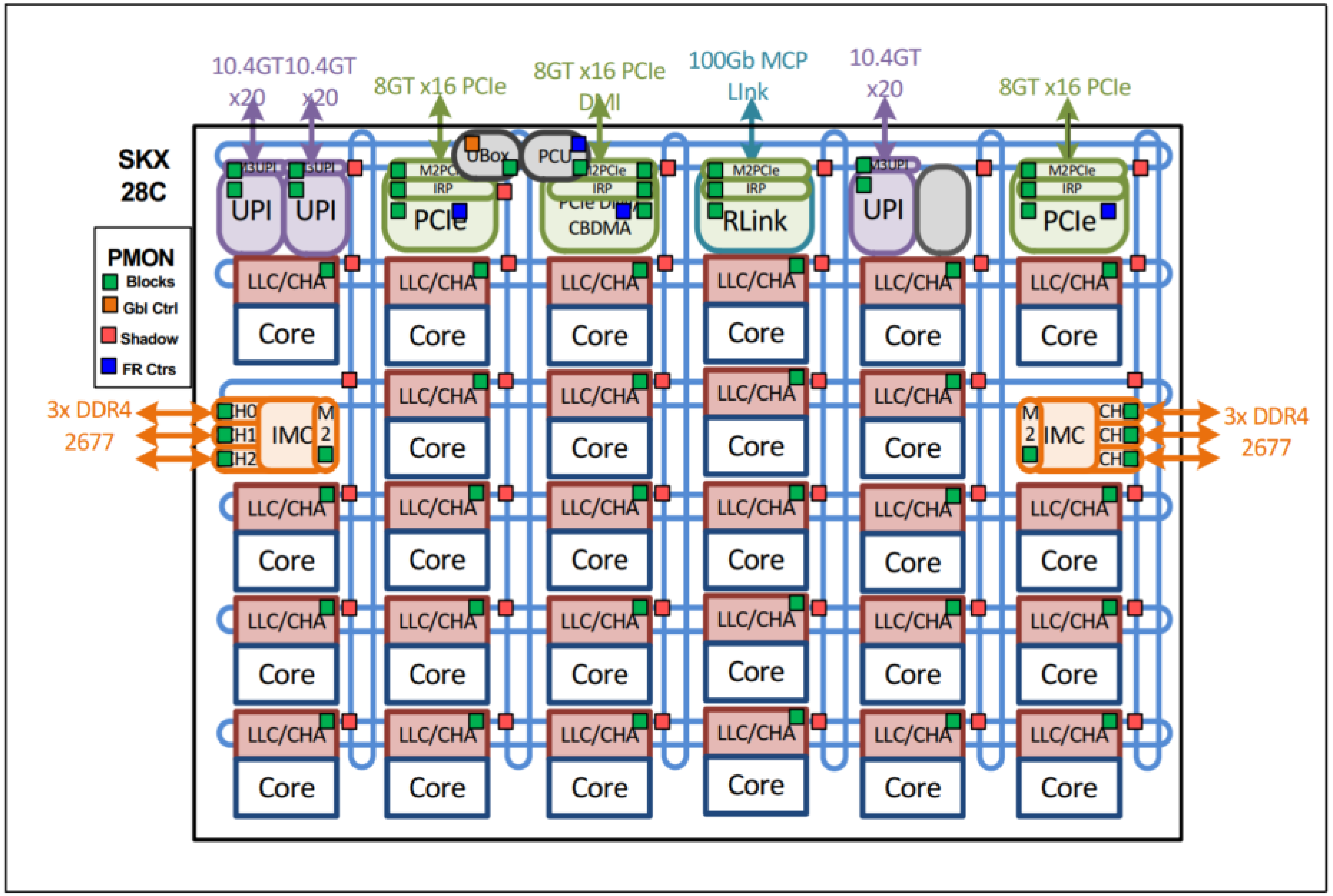
SKX uncore monitoring

David Levinthal

See uncore PMU pdf 336274

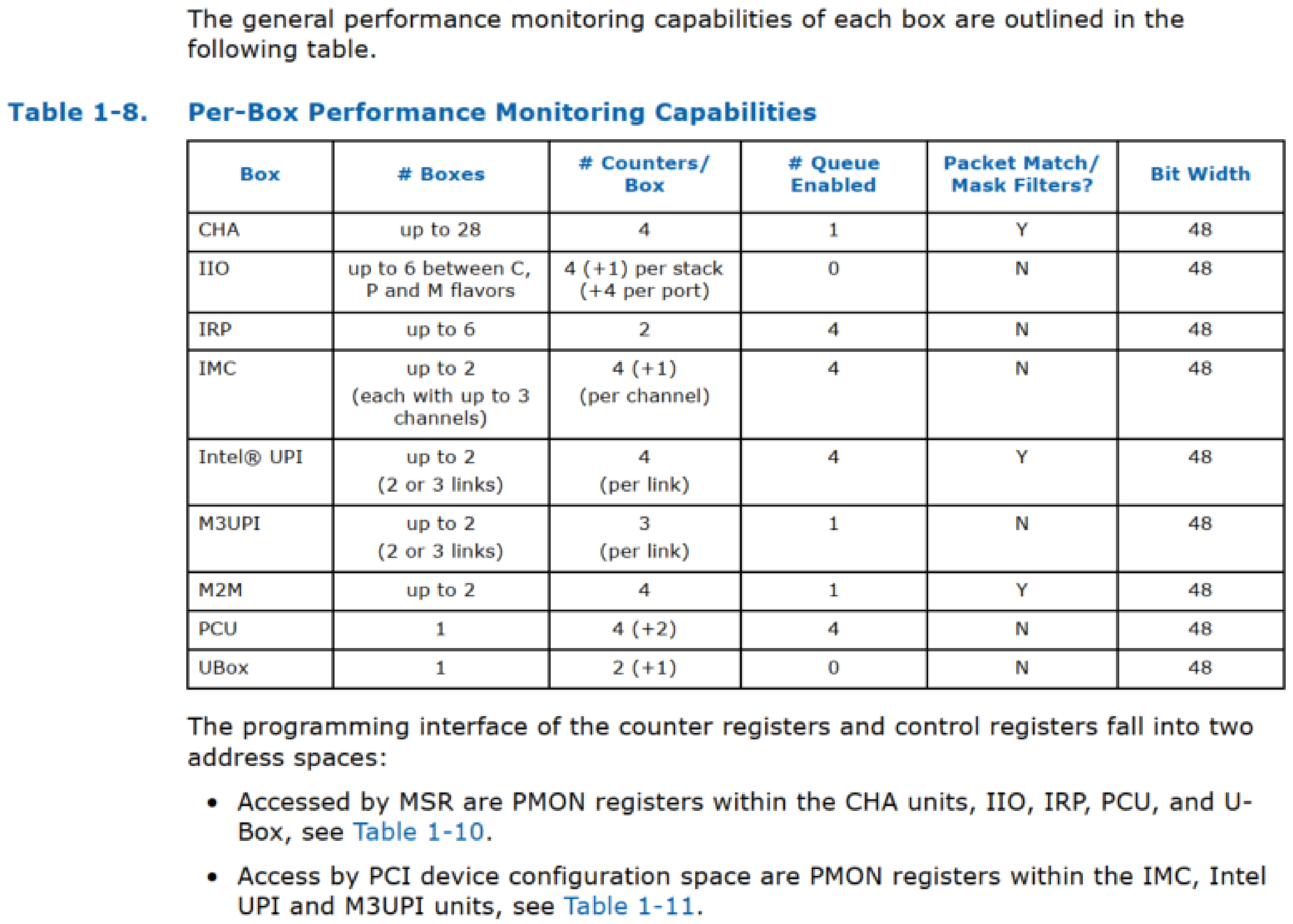
Figure 1-1. Intel® Xeon® Processor Scalable Memory Family - Block diagram for a 28C part



Basic Units

- Mesh made up of 2-dimensional structure of rings
  - 4 rings: AD(address) BL(Block/data) AK(Acknowledge) IV(Interupts)
- CMS common mesh stop (aka common ring stop)
  - Home agent and Cbox combined in SKX and called CHA
  - Cache box and core interact with CMS
  - Cache box: unit of L3 cache/mesh stop. Lines associated by hash encoding of physical address, so there is no relation between the Cbox and the core at a single mesh stop, other than the common mesh stop (CMS)
  - See sec 2.2.7.1
- IMC: Integrated Memory Controller
  - M2M mesh to memory controller traffic management
- UPI
  - M3UPI mesh to UPI link Layer traffic management
- PCIe
  - IIO/M2PCIe is mesh to PCIe traffic manager
  - IRP coherency in IIO traffic
- Ubox (system control)
- PCU (Power control Unit)

Uncore PMU resources



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

Table 1-8. Per-Box Performance Monitoring Capabilities

Box	# Boxes	# Counters/ Box	# Queue Enabled	Packet Match/ Mask Filters?	Bit Width
CHA	up to 28	4	1	Y	48
IIO	up to 6 between C, P and M flavors	4 (+1) per stack (+4 per port)	0	N	48
IRP	up to 6	2	4	N	48
IMC	up to 2 (each with up to 3 channels)	4 (+1) (per channel)	4	N	48
Intel® UPI	up to 2 (2 or 3 links)	4 (per link)	4	Y	48
M3UPI	up to 2 (2 or 3 links)	3 (per link)	1	N	48
M2M	up to 2	4	1	Y	48
PCU	1	4 (+2)	4	N	48
UBox	1	2 (+1)	0	N	48

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

- Accessed by MSR are PMON registers within the CHA units, IIO, IRP, PCU, and U-Box, see Table 1-10.
- Access by PCI device configuration space are PMON registers within the IMC, Intel UPI and M3UPI units, see Table 1-11.

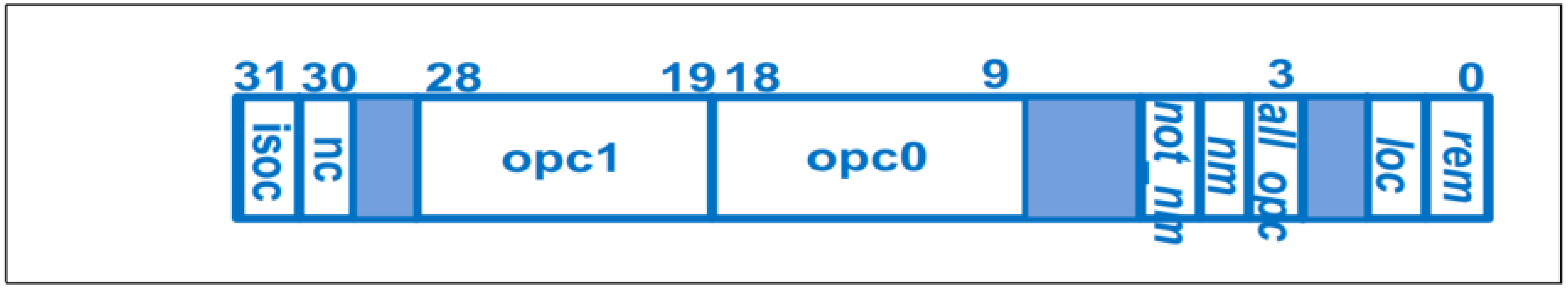
CMS terminology

- Traffic (2.2.7.1)
  - Rx receive
  - Tx transmit
  - RxC receive from CMS
  - RxR receive from ring
  - VN0/1 virtual channel 0/1
- IA: core
- Queues (2.2.7.2)
  - IRQ: request from core   (.IA events that were .IRQ events)
  - IPQ: ingress probe/remote socket snoop from UPI LL
  - ISMQ: response queue
  - PRQ: IIO request   (.IO events that were .PRQ events)
  - RRQ: remote read queue
  - WBQ: writeback queue
- TOR: Table of Requests, tracking CHA transactions

CHA event programming

- Basic event programming similar to core event programming
  - Event code, umask, threshold, invert, edge, etc
  - Emon threshold syntax is ":t=value"
- Two additional filters can be applied
  - Emon syntax is ":filter1=hexcode" or ":filter0=hexcode"
  - filter0 selects LLC mesi states (table 2-54) for LLC lookup events
  - filter1 selects request opcode (OR of up to 2 opc), local/remote etc
    - TOR_occupancy/TOR_inserts
    - Bottom byte = 0x33 is local or remote/opcode filter enable/cacheable or non cacheable

Figure 2-4. CHA PMON Filter 1 Register



CHA events
home agent, Core and CMS combined

- 1199 total events Some duplication to support old style home agent names
- TOR inserts and occupancies by request type / opcode matching
- Requests by MESI, Local, Remote, reads, writes, snoops, cross core snoops
- Directory lookups
- CMS interaction by receive, transmit
  - IPQ, RRQ, WBQ, PRQ
- LLC victimization
- IMC writes (memory BW per core?)
- Vert/Horiz traffic breakdown

TOR Occupancy LLC miss event examples

- Note: these are monitoring the cache box requests from all cores not the core at the CMS
- UNC_CHA_TOR_OCCUPANCY.IA_MISS:filter1=0x40433
  - 0x33 is is local or remote/opcode filter enable/cacheable or non cacheable
  - 0x40400 is actually 202 in bit 18:9 (opc0)
    - Corresponds to demand data rd (see table 3.1)
  - OPC1 is not used
  - Counts TOR occupancy of demand data rd that miss the LLC
- UNC_CHA_TOR_OCCUPANCY.IA_MISS:filter1=0x12D40433
  - 0x33 is is local or remote/opcode filter enable/cacheable or non cacheable
  - 0x40400 is demand data rd in opc0
  - 0x12D00000 is 0x25A in bits 28:19   ie opc1
    - LLC data prefetch (from 3-1 sheet 2)
  - Counts occupancy of demand data rd OR   prefetch data rd
    - ie all data requests from the core that miss the LLC

Tor Occupancy LLC miss examples cont.

- UNC_CHA_TOR_OCCUPANCY.IA_MISS:filter1=0x12CC0233
  - 0x33 is is local or remote/opcode filter enable/cacheable or non cacheable
  - 0x12CC0200 is
    - 0x202 in opc0 or demand code rd from table 3-1
    - 0x259 in opc1   or LLcPrefCode   from table 3-1 sheet 2
  - Counts demand code rd OR LLC prefetch code
    - ie all code requests from the core that miss the LLC
- UNC_CHA_TOR_OCCUPANCY.IA_MISS:filter1=0x12C40033
  - 0x12C40000 is
    - 0x200 in opc0 demand data RFO
    - 0x258 in opc1 LLcPrefRFO
  - Counts demand RFO OR LLC prefetch RFO
    - ie all reads for ownership   exclusive state that miss the LLC

Tor Occupancy LLC miss examples cont.

- UNC_CHA_TOR_OCCUPANCY.IA_MISS:filter1=0x12C40031
  - Same as before (ie all RFO)
  - Except 0x31 is remote target only, cacheable, non cacheable, opc enabled
  - Ie LLC RFO miss remote source

IMC (416 events)

- Integrated memory controller can monitor traffic to and from dram
- Most basic events are total line reads and writes
  - UNC_M_CAS_COUNT.RD
  - UNC_M_CAS_COUNT.WR
- And the read and write queue occupancy and queue inserts
  - UNC_M_RPQ_INSERTS
  - UNC_M_RPQ_OCCUPANCY
  - UNC_M_WPQ_INSERTS
  - UNC_M_WPQ_OCCUPANCY
  - Note: occupancy is evaluated at IMC clockticks

M2M IMC to mesh (457 events)

- Monitoring multi socket directory
  - UNC_M2M_DIRECTORY_LOOKUP.*
  - UNC_M2M_DIRECTORY_UPDATE.*
  - UNC_M2M_DIRECTORY_HIT.*
  - UNC_M2M_DIRECTORY_MISS.*
- Tracker activity
- WPQ activity
- Vertical and horizontal ring activity by ring
- 419 experimental events, 38 standard events

UPI link layer (158 events)

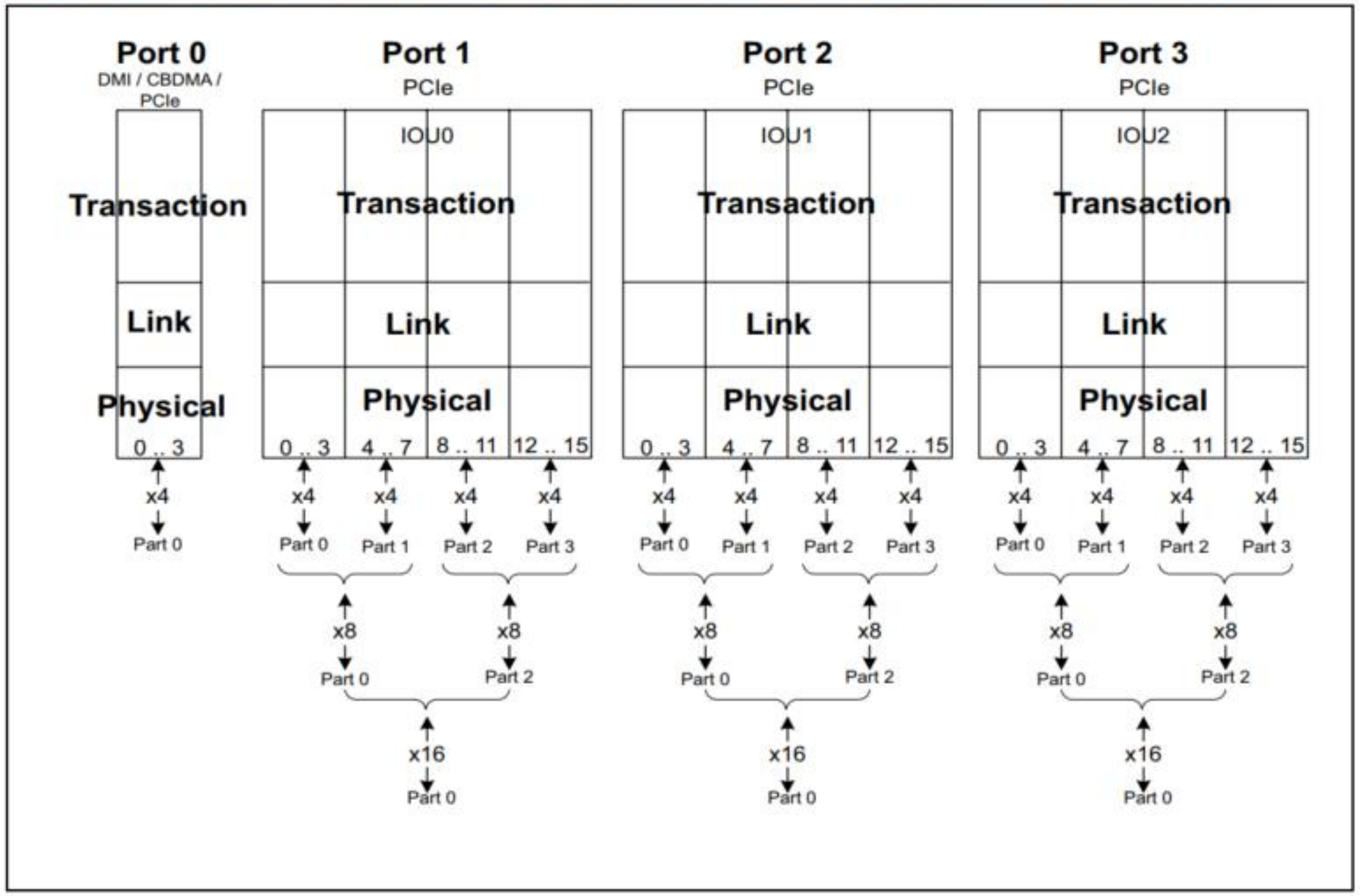
- Intersocket link layer traffic monitored (QPILL/UPILL)
    - Interaction with mesh monitored by m3upi
- UPI BW
    - UNC_UPI_RxL_FLITS.ALL_DATA (received flits)
    - UNC_UPI_TxL_FLITS.DATA (transmitted flits)
    - 9 flits needed to move a $line so multiply by 64/9 to get bytes
- Link in various power mode
    - examples shows transmit, similar expression for receive
    - UNC_UPI_RxL0_POWER_CYCLES/ UNC_UPI_CLOCKTICKS (full power)
    - UNC_UPI_TxL0P_POWER_CYCLES/ UNC_UPI_CLOCKTICKS (low power)
    - UNC_UPI_L1_POWER_CYCLES/ UNC_UPI_CLOCKTICKS (shutdown Rx and Tx)
- Inserts/Occupancy per UPI slot (latencies)
- UPI headers opcode matching
- Cycles with no credits in flowQ (not quite sure what these are
- Blocked receive queue cycles by cause

M3UPI Mesh-UPI interface (629 events)

- Transmit and receive flowQ occupancies and inserts for VN0/1
  - Request, snoop, response and WB for AD and BL Rings
- Cycles FlowQ not empty
- Arbitration events for Tx, for AD and BL
  - Failed, Speculative arb for new, no credit, credit available
- Arbitration events for Rx, for AD and BL
  - “lost” and “can’t” arbitrate
- Breakdown by Vert and Horiz by ring of transmit (and some ring to ring)
  - Occupancy, inserts, cycles_full

PCIe and IIO monitoring structure

Figure 2-8. How IIO BW/TXNs are rolled up relative to width of attached Device(s)

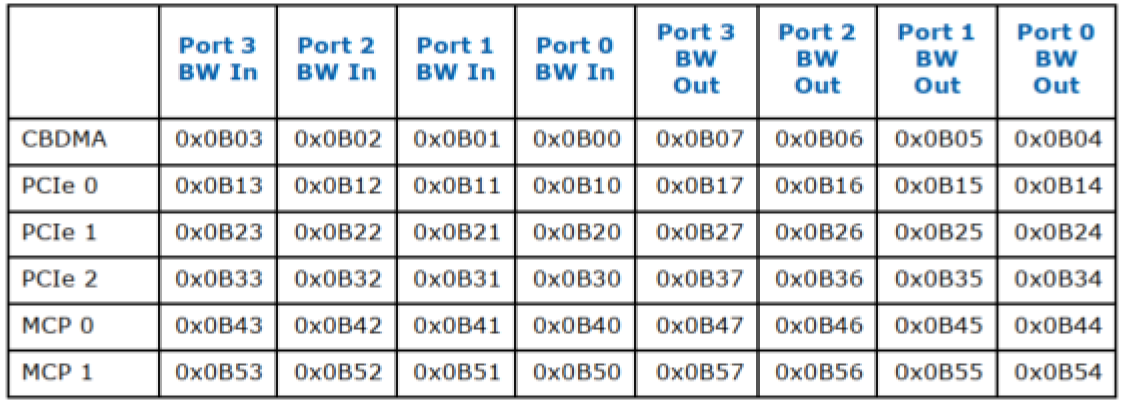


Events such as DATA_REQ_OF_CPU.MEM_READ and TXN_REQ_BY_CPU.PEER_WRITE are further subdivided into '.PART's. For instance, if this is a IIO stack supporting up to x16 PCIe and a x16 Card is plugged into the stack, .PART0 will contain all the relevant information. If 2 x8 Cards are plugged into the stack, .PART0 will contain the contribution of the first x8 Card's traffic and .PART2 will contain the contribution of the second x8 Card's traffic. etc.

PCIe and IIO monitoring with free running MSRs

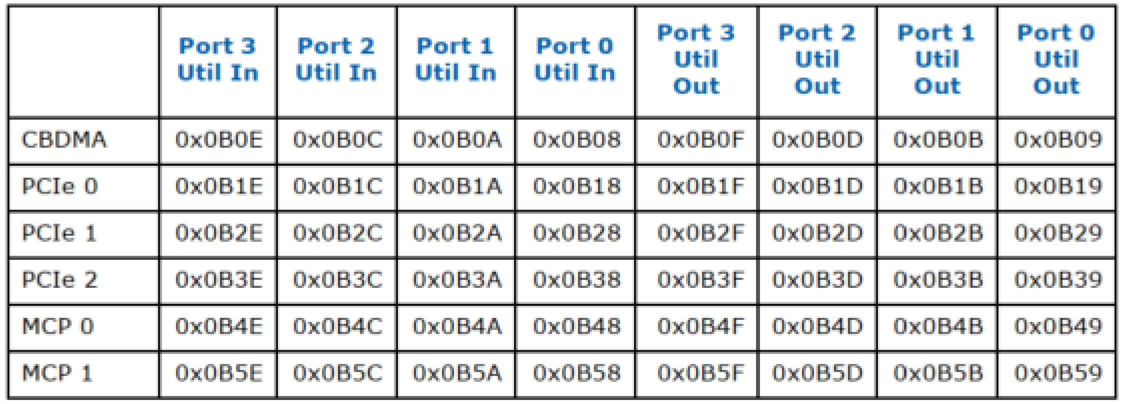
- The PCIe monitoring has a unique feature of monitoring the input and output bandwidth in free running counters grouped by 4 lanes

Table 1-11. Free-Running IIO Bandwidth Counters in MSR space



	Port 3 BW In	Port 2 BW In	Port 1 BW In	Port 0 BW In	Port 3 BW Out	Port 2 BW Out	Port 1 BW Out	Port 0 BW Out
CBDMA	0x0B03	0x0B02	0x0B01	0x0B00	0x0B07	0x0B06	0x0B05	0x0B04
PCIe 0	0x0B13	0x0B12	0x0B11	0x0B10	0x0B17	0x0B16	0x0B15	0x0B14
PCIe 1	0x0B23	0x0B22	0x0B21	0x0B20	0x0B27	0x0B26	0x0B25	0x0B24
PCIe 2	0x0B33	0x0B32	0x0B31	0x0B30	0x0B37	0x0B36	0x0B35	0x0B34
MCP 0	0x0B43	0x0B42	0x0B41	0x0B40	0x0B47	0x0B46	0x0B45	0x0B44
MCP 1	0x0B53	0x0B52	0x0B51	0x0B50	0x0B57	0x0B56	0x0B55	0x0B54

Table 1-12. Free-running IIO Utilization Counters in MSR space



	Port 3 Util In	Port 2 Util In	Port 1 Util In	Port 0 Util In	Port 3 Util Out	Port 2 Util Out	Port 1 Util Out	Port 0 Util Out
CBDMA	0x0B0E	0x0B0C	0x0B0A	0x0B08	0x0B0F	0x0B0D	0x0B0B	0x0B09
PCIe 0	0x0B1E	0x0B1C	0x0B1A	0x0B18	0x0B1F	0x0B1D	0x0B1B	0x0B19
PCIe 1	0x0B2E	0x0B2C	0x0B2A	0x0B28	0x0B2F	0x0B2D	0x0B2B	0x0B29
PCIe 2	0x0B3E	0x0B3C	0x0B3A	0x0B38	0x0B3F	0x0B3D	0x0B3B	0x0B39
MCP 0	0x0B4E	0x0B4C	0x0B4A	0x0B48	0x0B4F	0x0B4D	0x0B4B	0x0B49
MCP 1	0x0B5E	0x0B5C	0x0B5A	0x0B58	0x0B5F	0x0B5D	0x0B5B	0x0B59

PCIe and IIO monitoring with PMU

- Dominant IIO events are defined as matrices
- Event codes 83,84,c1,c2
  - Data_req_of/by_cpu, TXN_req_of/by_cpu
    - "Of" or "by" selects send vs receive
    - Data or TXN selects count of dwords or transaction requests
  - Read vs write
  - 4 "targets"
    - Memory (MMIO or dram depending on "by_cpu" vs "of_cpu" event type)
    - Peer (peer to peer)
    - IO (pcie IO space)
    - CFG (PCIeCFG space
  - 6 PCIe partitions
    - part0-3
    - VTD0-1
  - Data_req_of_cpu was called UNC_IIO_PAYLOAD_BYTES_IN
    - IO to IO transfer

Measuring memory bandwidth

- Memory bandwidth is a time average count and can be measured at the memory controller
  - Read bw = 64*UNC_M_CAS_COUNT.RD/UNC_M_CLOCKTICKS
  - Write bw = 64*UNC_M_CAS_COUNT.WR/UNC_M_CLOCKTICKS
  - total bw = 64*UNC_M_CAS_COUNT.ALL/UNC_M_CLOCKTICKS

Occupancy and insert events

- The ratio of occupancy/inserts is an average latency or time in the queue (aka Little's Law)
- There are 37 occupancy events with multiple umasks in the event files
    - Some support setting multiple umask bits
        - Result is the occupancy/inserts for the 'OR' of the umasks
    - Some do not
        - 'OR of umasks does not produce occupancy/inserts of the OR
    - Occupancy events that only have hardcoded umasks can't be ORed. ('00001000', '00000010', etc)
    - Occupancy events that use the common notation where each subevent has it's own bit with don't cares on the other bits. ('xxxx1xxxx', 'xxxxxx1x', etc)