Field-programmable Port eXtender (FPX) Support for the Network Services Platform (NSP)

Version 1.0

Alex Chandra, Yuhua Chen, John DeHart, Sarang Dharmapurikar,
Fred Kuhns, John W. Lockwood, Wen-Jing Tang, David E. Taylor,
Jonathan S. Turner

WUCS-TM-02-??

September 23, 2005

Applied Research Laboratory
Department of Computer Science
Washington University in Saint Louis
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130

Abstract

This document describes the design and functionality of the hardware components implemented in the Field-programmable Port eXtender (FPX) to support the Washington University Network Services Platform (NSP). This includes support for the Multi-Service Router (MSR) and Extreme Networking projects. The functionality of each component is described along with supporting top-level entity diagrams, block diagrams, and interface timing diagrams. For more information on the NSP and related projects, see [1][2]. For more information on the FPX, see [3][4][5][6][7].

Supported by: NSF ANI-0096052

Contents

1 Overview

The Network Services Platform (NSP) [1] project seeks to build an open-platform programmable router for networking research. Building upon prior research, the NSP uses the Washington University Gigabit Switch (WUGS) [8], an ATM switch configured with eight ports each supporting 2.5 Gb/s links, as the core switching fabric. Exceptional and "active" packet processing is handled by the Smart Port Card (SPC) [2], a port card containing an embedded microprocessor and custom network interface device. The Field-programmable Port eXtender (FPX) [3][4] handles a majority of the "plain" packet processing and queuing functionality. A logical block diagram of the NSP with FPX support is shown in Figure 1.

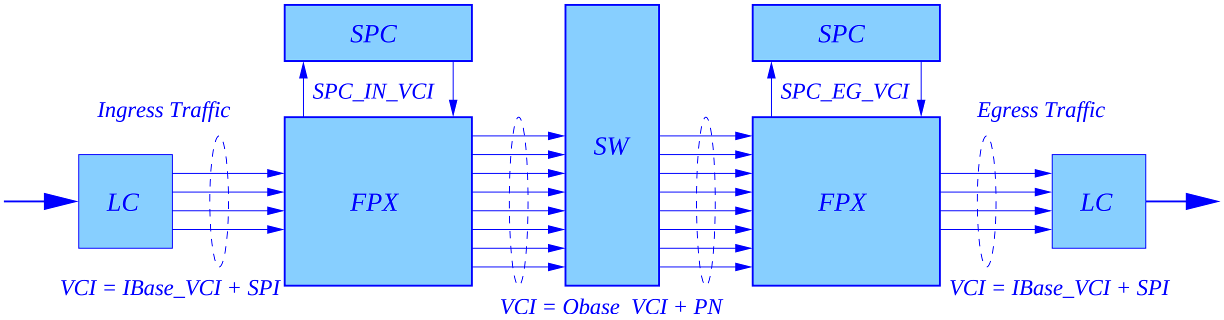


Figure 1: $overview_{m}sr$ Logical block diagram of the Network Services Platform (NSP) with FPX support. Both input and output VCI mappings use a base plus offset scheme to provide for ease in configuration.

The NSP supports four virtual interfaces per port identified by a unique Virtual Circuit Identifier (VCI) equal to an input base VCI (IBase_VCI) plus sub-port identifier (SPI). Packets are forwarded to the appropriate output port using a port-specific VCI equal to an output base VCI (OBase_VCI) plus the port number (PN). Packets requiring processing by the SPC are transmitted to and received from the SPC on one of two VCIs (SPC_IN_VCI or SPC_EG_VCI). The need for distinct ingress (IN) and egress (EG) VCIs will be explained in Section 2.2.

A logical block diagram of the hardware components of the NSP is shown in Figure 2. Note that both ingress and egress traffic share a single datapath. All traffic arrives at the Input Segmentation and Reassembly (ISAR) block in the form of AAL5 encapsulated frames. The ISAR extracts the payload of the ATM cells of the AAL5 frames, which includes the Internet and Transport protocol headers. Note that cells of frames may become interleaved among the four virtual interface VCIs, the eight switch port VCIs, and two SPC VCIs; therefore, the ISAR must maintain a total of 14 reassembly contexts. For traffic arriving from the four virtual interface VCIs, the ISAR must insert an NSP Shim, an internal packet header used to communicate information regarding packet handling throughout the NSP. The contents and functionality of the shim are explained in Section 3.2. The ISAR buffers then writes fixed size chunks of arriving packets to the Packet Storage Manager (PSM).

Once the entire packet has been received in the ISAR, the packet pointer, shim fields, and packet header fields are forwarded to the Classification and Route Lookup (CARL) block for classification. Upon completion of a lookup, CARL updates the shim fields and, if necessary, makes copies of the packet header fields. This will occur in the case of multicast packets or a non-exclusive filter match. Note that only one copy of the packet is stored in SDRAM, while multiple copies of the packet header may exist.

The packet pointer, packet length, copy count, and shim fields are sent to the Queue Manager (QMGR). Based on the shim fields, the Queue Manager decides which queue to place the packet on. The Queue Manager schedules packet transmission according to parameters for each type of queue. Parameters range

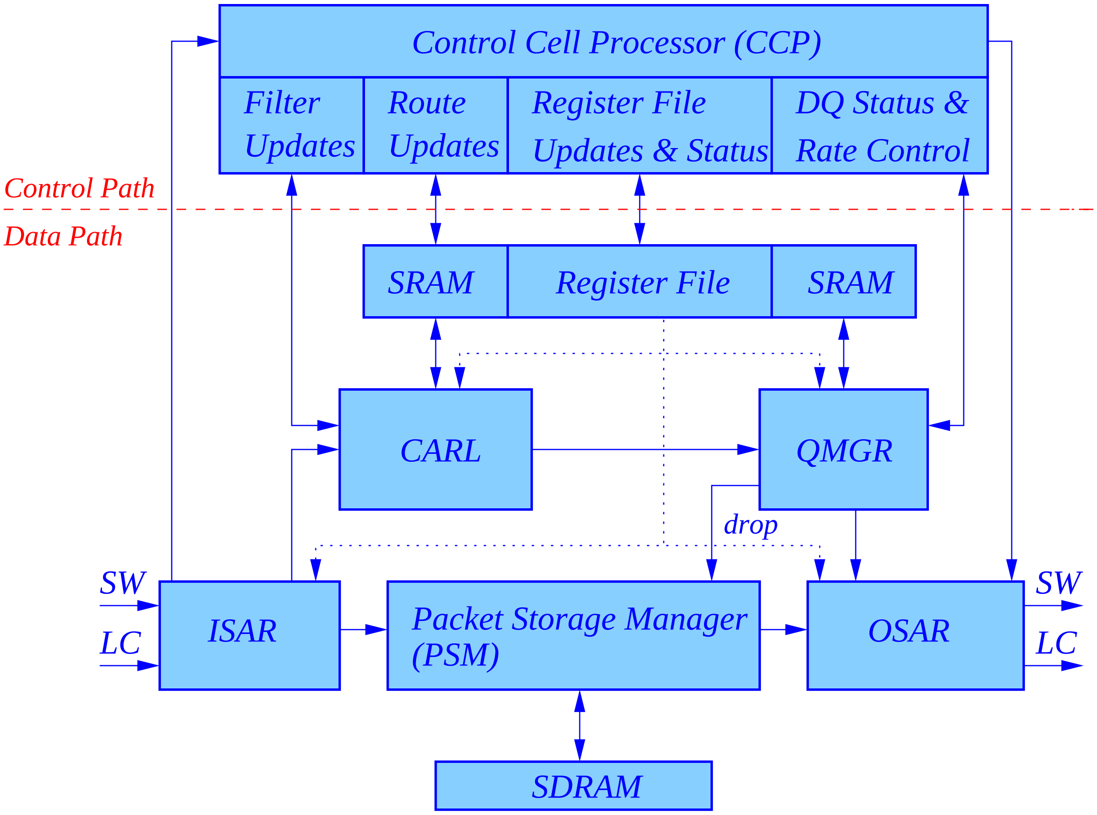


Figure 2: $overview_block$ Logical block diagram of the components in the Reprogrammable Application Device (RAD) of the Field-programmable Port eXtender (FPX) for the Network Services Platform (NSP).

from packet length to rates distributed between ports in control cells received by the CCP.

The packet pointer, shim fields, and copy count of the next packet to be sent is forwarded to the OSAR. The OSAR retrieves the packet from the PSM, creates an AAL5 frame, and transmits the cells of the frame on either the RAD switch or line card interface. Note that cells of a frame cannot be distributed across the two interfaces to prevent reordering. The OSAR must also remove NSP Shims from the header of packets transmitted on one of the four virtual interfaces (LC). For packets with copy counts greater than one, the PSM must keep track of how many copies of the packet have been sent.

Note that control cells are switched to the control path at the input of the ISAR and processed by the Control Cell Processor (CCP). The CCP is responsible for managing the packet classification database, register file, and queuing parameters. Response cells are switched into the datapath at the outputs of the OSAR block.

2 Design Constraints

It is important to understand the constraints either imposed by the line rate or by the implemented FPX board. These constraints are identified and reviewed in this section. The rest of the document describes the functionality of the hardware components implemented in the Reprogrammable Application Device (RAD) of the FPX in order to support the NSP.

2.1 Performance Constraints

The goal of this project is to support 1Gb/s links. With a 2x speedup in the switch fabric along with 200Mb/s of traffic to and from the SPC, the total uni-directional bandwidth is 3.2Gb/s. Due to previous studies performed on the Xilinx Virtex-E 2000 (xcv2000e-6), a target clock frequency of 75MHz was chosen.

The target link rate requires that total buffer I/O be 6.4Gb/s (approximately 15M packets per second) to support this system. The combined raw throughput of the SDRAM interfaces of the RAD operating at 75MHz is 9.6Gb/s. Therefore, the PSM must sustain an average efficiency of 67% on the SDRAM interfaces. The raw throughput provided by the SRAM interface used by Queue Manager is 2.7Gb/s. This allows an average of five (5) memory transactions per packet.

For classification, the General Filter Match and the Route Lookup blocks must only operate on traffic arriving from the line card. Therefore, they must sustain a throughput of 1Gb/s (2.36M lookups per second). However, the Exact Match classifier must operate on traffic arriving from the switch and the line card (we assume that re-classification traffic from the SPC is negligable). Therefore, it must sustain a throughput of 3Gb/s (7M lookups per second). The Route Lookup and Exact Match classifier must share a single SRAM interface with a raw bandwidth of 2.7Gb/s.

2.2 Physical Design Mapping

Due to the physical layout of the FPX board and subsequent pin-mappings on the RAD, the logical block diagram requires a few changes in order to physically map to the RAD. The physical block diagram is shown in Figure 3.

Note that the PSM is divided into two independent homogeneous blocks due to the location of the two SDRAM interfaces. The physical split of the PSM prevents excessive capacitive loading and exhaustion of long-line routing resources in the RAD. Likewise, the ISAR and OSAR will act as two pseudo-independent blocks. The ISAR has independent input packet processing sub-blocks with associated PSM interfaces. However, the sub-blocks are not completely independent as the ISAR must multiplex packet headers for transmission on the single interface to CARL. Similarly, OSAR has independent output packet processing sub-blocks with associated PSM interfaces. However, OSAR must de-multiplex packet headers arriving from the Queue Manager to the correct sub-block and switch output packets to the correct NID/RAD interface for transmission to the switch or line card. This final switching step is necessary to prevent interleving cells of different packets bound for the same destination.

The CCP receives control cells from the ISAR and transmits response cells to the OSAR. The CCP maintains a global register file, the routing table for CARL in SRAM, as well as handling the distributed rate control information for the Queue Manager. All RAD control cells are handled by the CCP.

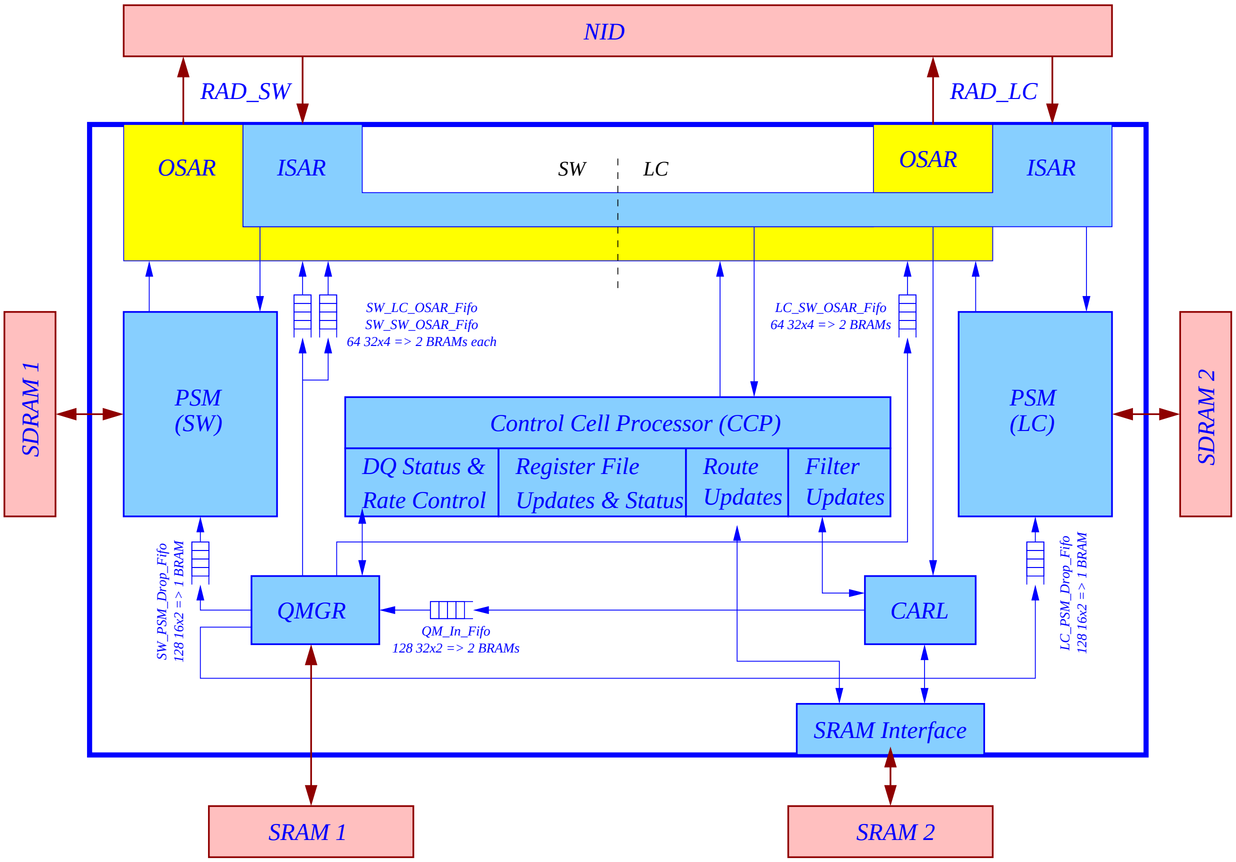


Figure 3: $overview_block_phy$ Block diagram of the components in the Reprogrammable Application Device (RAD) of the Field-programmable Port eXtender (FPX) for the Network Services Platform (NSP) depicting the physical mapping constraints of the FPX board.

2.3 NID Interface

In order to route traffic to the RAD, the appropriate entries must be written to the Virtual Circuit Translation Table (VCXT) of the Network Interface Device (NID) of the FPX. A diagram of the NID VCI routing is shown in Figure 4; note that the specific value of the entries in the VCXT will depend on the configuration of the NSP. Users should see [9] regarding the operation and programming of the NID.

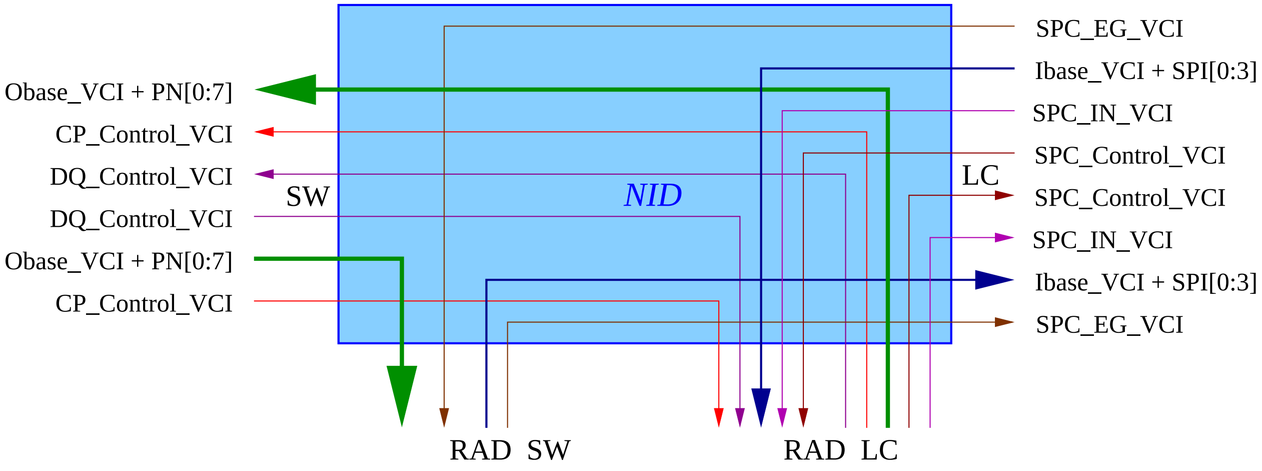


Figure 4: nid_vci_route Virtual Circuit Identifier (VCI) routing in the Network Interface Device (NID) of the Field-programmable Port eXtender (FPX) for the Network Services Platform (NSP).

The following constraints to NID were added during May 22, 2002 meeting.

- The NID status cell should allow reporting of the UP_LC_LINK signal from the line card. This is just another bit that should appear in the NID status cell. Note that the NID currently reports UP_SW_NID as always up to the switch. No change for this. We should check that UP_SW_NID reports that the links is down during reset.

- We need to assume that line cards do not have clean UP_LC_LINK signals. Whenever the UP_LC_LINK signal report that the link is down, a timer/state machine should ensure that no cells are sent or received to/from the line card for 100ms on the SW_CLK (assuming 62.5 MHz).

  Add link-enable state machine to NID. The state machines adds 100 ms delay after carrier present before link is in UP state. Reset puts the state machine in link down state. NID should ignore cells from the link if it is not in UP state.

- A new NID control cell should be added that allows a RAD_RESET to be issued via a control cell.

- Add maintenance register and control cells processing in NID so that switch can read link states.

- Control cell should be able to poll to check hardware status.

- NID assserts link-up signals to switch when NID is up (after NID) is ready to receive control cells). Notes: NID does not initialize VXT table at reset. VXT table is only initialized after power up.

- NID verifies HEC value of all ATM cells bound to RAD, dropping errornous ones.

- NID needs to be able to forward one cell every 16 cycles per interface. This translate to 2.0Gbps.

2.4 Programming of RAD

RAD needs to be programmed after SPC is up. If SPC is reloaded, RAD will be reprogrammed.

2.5 Reset and Initialization

RAD needs to have auto reset circuitry after it is programmed to force proper initialization

Every RAD submodule generates READY signal to ISAR after initialization. ISAR outputs nothing and ignores incoming cells until all modules are ready. ISAR only sends READY signal to NID after all modules are ready.

2.6 MTU

The MTU for NSP system is 2000 bytes. This is the largest IP packet, not including shim and AAL5 trailer, being processed by FPX. IP packet larger than this will be discarded by ISAR.

3 Inter Component Communication

A limited communication mechanism among components and sub-components of the NSP is embedded with the packet. The nature of this communication is forward looking; it is mainly used to carry additional information relevant to the packet such as its location in the SDRAM, intermediate results of packet processing performed, etc.

3.1 Flags

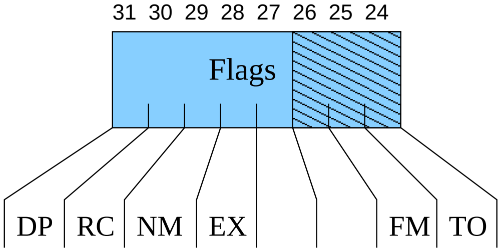


Figure 5: msr_shim_flags Format of the shim Flags field.

The Flags field, as shown in Figure 5, is used for passing packet handling information between the FPX and SPC. Note that the lower three flags (shaded in the figure) are FPX-specific state information which must be retained with the packet and must not be modified by the SPC. The flags are defined as follows:

- **DP (Drop Packet)**: This flag is set when a packet should be dropped due to failed checksums, mismatched length fields, filters, or SPC directives. Accordingly, this flag may be set by ISAR, CARL, QMGR, or the SPC.
- **RC (Re-Classify packet)**: This flag is set by the SPC when a packet requires re-classification in the FPX due to processing in the SPC.
- **NM (No Match)**: This flag is set by CARL when no filter or lookup entry exists for the packet. The packet is sent to the SPC for classification.
- **EX (Exception packet)**: This flag is set by ISAR when IP Options exist in the packet header or the packet is not IPv4. The packet is sent to the SPC for processing.
- **FM (From LC/SW)**: This flag allows components to distinguish the source of the packet via a single bit compare. 0 = From Switch, 1 = From Line Card. This flag is set by ISAR. The SPC must not modify this flag.
- **TO (To LC/SW)**: This flag allows components to distinguish the destination of the packet via a single bit compare. 0 = To Switch, 1 = To Line Card. This flag is set following priority resolution by CARL. The SPC must not modify this flag.

Inside the FPX, another set of flags is used for inter-module communication. As shown in Figure 6, the Internal Flags are an 8-bit field passed between components. The Internal Flags are defined as follows:

- **DG (Datagram packet)**: This flag is set by CARL when the only lookup result for the packet is a route entry.
- **SB (SPC-bound packet)**: This flag is set to 1 when the packet should be transmitted to the SPC. This flag is set by ISAR and CARL for exception packets, no match, etc.
- **SR (SPC-return packet)**: This flag is set to 1 when the packet is returning from the SPC. This flag is set by ISAR.

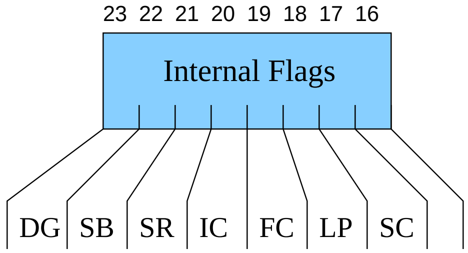


Figure 6: $msr_{s}him_{i}nt_{f}lags$ Format of the Internal Flags field.

- **IC (Initial Copy)**: This flag is set to 1 by the CARL when it sends the first copy of any packet to QMGR.
- **FC (Final Copy)**: This flag is set to 1 by the CARL when it sends the last copy of any packet to QMGR.
- **SC (Single-Chunk packet)**: This flag is set to 1 when the entire packet resides in one chunk of SDRAM. This flag is set by ISAR.

3.2 Shims

To pass packet handling information between functional blocks of the NSP, a custom header field, called a shim, is used. The shim is added to packets arriving from the links and deleted from packets transmitted on the link. The format of the InterPort Shim used to communicate between input and output ports is shown in Figure 7. The InterPort shim contains the following fields: Input VIN (Virtual Interface Number) and Output VIN (Virtual Interface Number).

The VIN (Virtual Interface Number) denotes the physical port and sub-port as shown in Figure 9. The sub-port denotes one of four Virtual Circuit Identifiers (VCIs) used for transmission on the link. The Input VIN is the physical port and sub-port at which the packet arrived from a line card. This field is stamped by the ISAR upon packet arrival. The Output VIN is the output port and sub-port of the packet returned by CARL.

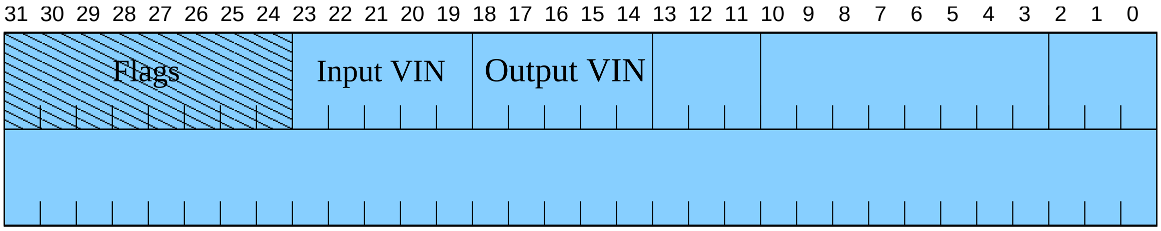


Figure 7: $msr_{i}nterport_{s}him_{f}ormat$ Format of the InterPort Shim used to communicate between input and output ports.

Note that presently there are no Flags defined for the InterPort shim, however we reserve the first octet of InterPort Shim to be used for such purpose.

The format of the IntraPort Shim used to communication between processing components of a single physical port (the SPC and the FPX) is shown in Figure 8. The IntraPort shim contains the following fields: Flags, Input VIN (Virtual Interface Number), Output VIN (Virtual Interface Number), Queue Identifier (QID), Total Chunks (TTL Chunks), Queue Length, Packet Pointer, and Second Chunk Pointer.

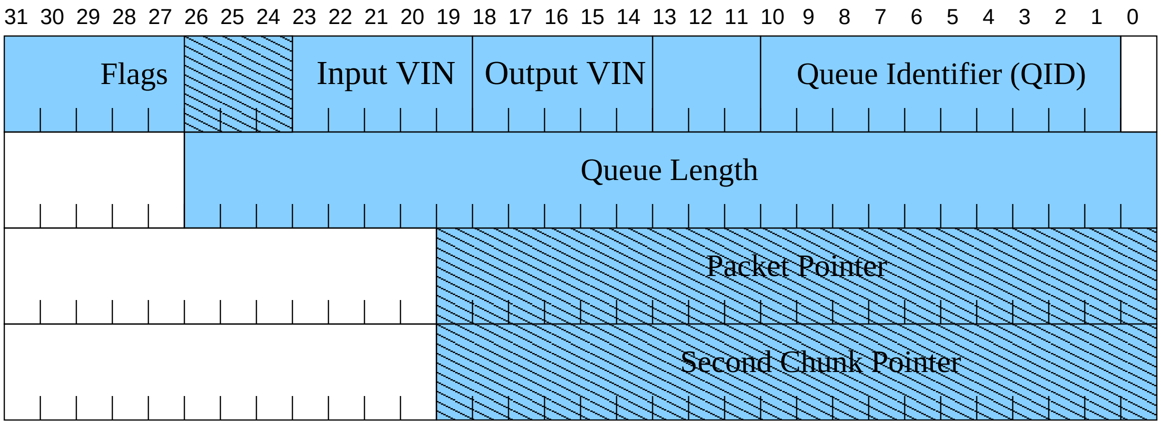


Figure 8: $msr_intraport_shim_format$ Format of the IntraPort Shim used to communicate between processing components of a single physical port.

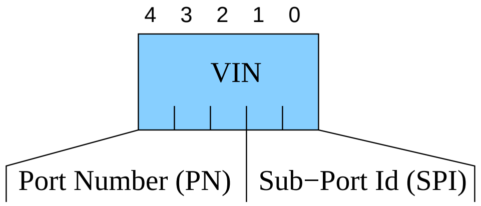


Figure 9: msr_shim_vin Format of the shim Virtual Interface Number (VIN) fields.

The Input VIN (Virtual Interface Number) and Output VIN (Virtual Interface Number) fields are identical to the InterPort Shim. The Queue Identifier (QID) is a local label that designates the queue for the packet. In the case of "active" and software reserved flows, this QID also acts as a stream identifier for flows requiring processing in the SPC. The TTL Chunks field denotes the number of 128 byte chunks of SDRAM used by the packet. This field must not be modified by the SPC, as it is necessary to manage memory in the FPX.

The Queue Length field reports the queue length in bytes associated with the QID. The Packet Pointer and the Second Chunk Pointer are vestiges of a feature no longer supported in the FPX and can be ignored.

4 System Functional Description

To clarify the packet processing steps, the following sub-sections enumerate the steps for each supported permutation.

4.1 Management Traffic

Control cells are used for managing the FPX system. The control cells are switched to the control path at the input of the ISAR and processed by the Control Cell Processor (CCP). The CCP is responsible for managing the packet classification database, register file, statistics, and queuing parameters. Response cells are switched into the datapath at the output of the OSAR block.

Based on the communicating entities, management traffic can be grouped into three classes. Between CP and CCP, Control cells initiated by CP must arrive at SW Interface with CP_Control_VCI Between SPC and CCP, Control cells coming from SPC will arrive at LC Interface on either SPC_Control_VCI, Between one CCP and another CCP to facilitate distributed queuing, QM_Control_VCI

The varying formats of these control cells are documented in Section 5.6.

4.2 Line Card Traffic

Packets arriving from the line-card on sub-ports SP0 through SP3 are switched to the RAD_LC port of the RAD by the NID. The ISAR sets the FM flag to 1 = Line Card and writes the Input VIN field based on the PN register in the register file. The ISAR packs the packet contents into chunks and forwards them to the PSM (LC). The PSM returns a unique packet pointer while the first chunk is forwarded. The ISAR is responsible for checking the IP header checksum, the AAL5 checksum, and verifying that the IP total length and AAL5 length match. If any of the checks fail, the drop packet flag of the shim is set.

Once the entire packet is received, the ISAR passes the packet fields necessary for classification and queuing to CARL as described in Section 6.3.

The results of the lookup for a unicast ingress packet will specify an output port and/or queue identifier (QID) for queuing. CARL updates the Output VIN and QID shim fields based on the results of the lookup. CARL forwards the packet pointer, shim fields, packet length, and Rates (for reserved flow packets) to the Queue Manager. Based on these fields, the Queue Manager will select a queue on which to place the packet.

When the Queue Manager schedules a packet for transmission, the packet pointer and shim fields are sent to the OSAR. Since the FM flag bit is set to 1 = Line Card in the shim, the OSAR demultiplexes the packet header to the sub-block interfacing to the PSM (LC). The OSAR issues the packet pointer to the PSM and receives the chunks of the packet. Note that if the drop flag is set, the Queue Manager places the packet pointer on a special drop queue to the PSM.

In order to transmit the packet, the OSAR must do the following:

- Configure the InterPort Shim; update the Output VIN
- Create an AAL5 frame containing the packet (fragment the packet into ATM cells, compute AAL5 checksum over entire packet, write checksum and packet length in AAL5 trailer)
- Map the VCI of all cells of the frame to the appropriate switch port based on the Output PN (OBase_VCI + PN)

Note that ingress traffic arriving from the line card will be sent into the switch; therefore, the OSAR must switch all cells of the AAL5 frame to the RAD_LC port and send them to the NID.

4.3 Switch Traffic

Packets arriving from the switch on ports PN0 through PN7 are switched to the RAD_SW port of the RAD by the NID. The ISAR sets the FM flag to 0 = Switch in the shim and packs the packet contents into chunks and forwards them to the PSM (SW). The PSM returns a unique packet pointer while the first chunk is forwarded. The ISAR is responsible for checking the IP header checksum, the AAL5 checksum, and verifying that the IP total length and AAL5 length match. If any of the checks fail, the drop packet flag of the shim is set.

Once the entire packet is received, the ISAR passes the packet fields necessary for classification and queuing to CARL as described in Section 6.3.

The results of the lookup for a unicast egress packet will specify an outgoing port (Output PN), sub-port identifier (SPI), and/or queue identifier (QID) for queuing. CARL updates the Output VIN and QID shim fields based on the results of the lookup. CARL forwards the packet pointer, shim fields, packet length, and Rates (for reserved flow packets) to the Queue Manager. Based on these fields, the Queue Manager will select a queue on which to place the packet.

When the Queue Manager schedules a packet for transmission, the packet pointer, and shim fields are sent to the OSAR. Since the FM flag bit is set to 0 (SW) in the shim, the OSAR demultiplexes the packet header to the sub-block interfacing to the PSM (SW). The OSAR issues the packet pointer to the PSM and receives the chunks of the packet. Note that if the drop flag is set, the OSAR asserts a drop signal with the packet pointer which prompts the PSM to free the associated packet chunks. In order to transmit the packet, the OSAR must do the following:

- Remove the shim
- Decrement the IPv4 TTL
- Update the IPv4 header checksum (using the incremental update algorithm)
- Create an AAL5 frame containing the packet (fragment the packet into ATM cells, compute AAL5 checksum over entire packet, write checksum and packet length in AAL5 trailer)
- Map the VCI of all cells of the frame to the appropriate sub-port based on the OutputPN (IBase_VCI + SPI)

Note that egress traffic arriving from the switch with the TO flag set to 1 (LC) will be sent to the line card; therefore, the OSAR must switch all cells of the AAL5 frame to the RAD_SW port and send them to the NID.

4.4 SPC Traffic: Active and Exception Processing

Packets arriving from the line card and switch may require processing by the SPC for many reasons: active processing, unsupported protocols or options). Due to the physical partitioning of the PSM, there must be two unique VCIs in order to prevent interleaving of packet cells and direct returning SPC traffic to the correct ISAR/PSM pair.

When the OSAR is sends a packet to the SPC it retrieves the entire packet from the PSM and performs the following steps:

- Configure the IntraPort shim; update the Output VIN, flags, QID, Queue Length, TTL Chunks, and packet pointers.
- Create an AAL5 frame containing the packet (fragment the packet into ATM cells, compute AAL5 checksum over entire packet, write checksum and packet length in AAL5 trailer).
- Map the VCI of all cells of the frame based on the transmission port (see discussion below).

Instead of using the Output VIN for VCI mapping, packets transmitted on the RAD_SW port use SPC_EG_VCI, while packets transmitted on the RAD_LC port use SPC_IN_VCI. The transmission port is based on which PSM retrieves the packet from. Full packets returning from the SPC will handled by the ISARs in the normal manner. Note that the RC (Re-Classify) bit may be set by the SPC to signal CARL to re-classify the packet. The SPC may also decide to drop a packet by setting the drop flag (DP).

4.5 Multiple Copies of Packets

Many applications require multiple copies of a packet, for example, network monitoring. The copy count of a packet is set by CARL based on the results of classification. In the case of network monitoring, one non-exclusive filter may match the packet and require an additional copy of the packet to be sent to applications in the SPC on specific queue identifiers (QIDs). Based on the lookup results, CARL makes the required copies of the packet header fields, updates the necessary shim fields and copy count, and sends the header fields to the Queue Manager. The Queue Manager maintains the copy count with the packet header fields in SRAM.

Once a multi-copy packet header is received at the OSAR (denoted by the Final Copy FC flag = 0), the OSAR initiates a multi-copy transaction with the PSM. Once the copy count reaches zero, the QMGR sets the FC flag in the shim of the last packet scheduled for transmission. Upon receipt by OSAR and retrieval from the PSM, the packet chunks are freed in memory. OSAR takes the appropriate transmission actions based on the destination of the packet.

4.6 Specific Traffics

4.6.1 Multicast Traffic

Note that multicast is a partially implemented feature and should not be relied on without consulting the designers.

4.6.2 Network Monitoring Traffic

4.6.3 Broadcast Traffic

4.6.4 Software Reserved Traffic

4.6.5 Datagram Traffic

4.6.6 TTL=1 or TTL=0 Traffic

IP packets arriving at the LC interface with TTL=1 or TTL=0 will be forwarded to SPC.

4.6.7 ICMP Traffic

ICMP packet will be forwarded to SPC.

4.6.8 Fragmented IP Traffic

Processing of fragmented IP packets will not be supported. These packets will be dropped by ISAR and per-VCI packet drop counter will be incremented.

5 Component Functional Description

In this section, the function of each component of NSP-RAD will be described.

5.1 Input Segmentation and Reassembly (ISAR)

Top level signals of the Input Segmentation and Reassembly (ISAR) module is shown in Figure 10. ISAR receives IP packets which are encapsulated in AAL5 frames on both the line card (LC) and switch (SW) interfaces. Section 6.1 describes the LC and SW interface including the AAL5 frame format and packet contents. The AAL5 frame is carried in one or more ATM cells. HEC field of each arriving ATM cell is not verified. It is assumed that NID will drop cells with corrupted header prior to passing them to ISAR.

On SW interface ISAR may receive control cells sent by the Software. On LC interface, ISAR may receive control cells from SPC. Both control cells should be merged and passed to CCP.

ISAR interfaces with both LC (Ingress) PSM and SW (Egress) PSM. Packets arriving on the LC interface will be forwarded to the LC PSM, while those arriving on the SW interface to the SW PSM. ISAR is capable to support simultaneous processing of two packets one coming from LC and another from SW interfaces. Generated packet headers will be multiplexed before forwarded to CARL.

Packets arriving on LC interface may come from either the line card (VCI = IBase_VCI + SPI) or the Smart Port Card (SPC) (VCI = SPC_IN_VCI). Non SPC packets are formated into one or more 128-byte chunks, organized as in Figure 53 and Figure 54. For the first chunk, local Input VIN is stamped on the IVIN field. These chunks are then forwarded to LC PSM.

Pakets arriving on SW interface may come from either the switch (VCI = OBase_VCI + PN) or the SPC (VCI = SPC_EG_VCI). Non SPC packets have an 8-byte *interport* shim prepended to each packet as depicted in Figure 49. As on the ingress case, these packets are also formated into one or more 128-byte chunks. This time the IVIN value specified in the shim are used.

CP_Control_VCI is used for the Control Processor (CP) software to communicate with CCP. This channel is only defined on the SW interface. SPC_Control_VCI is used for the SPC control software to communicate with CCP. This channel is only defined on the LC interface. DQ_Control_VCI is used for QMGR to communicate the with other QMGR on the same NSP. This channel is only defined on the SW interface. Cells arriving with any of these VCI values at appropriate interface will be switched to CPP, otherwise they are dropped.

Packets arriving from SPC (VCI = SPC_IN_VCI or SPC_EG_VCI) have a 16-byte *intraport* shim is prepended to each packet as shown in Figure 50.

For each packet arrives at ISAR, a packet header (reference) as defined in Figure 56 is created and forwarded to CARL.

Flags fields are to be defined by ISAR as follows.

- **DP** is set to '0', unless any of the following condition occurs in which case it is set to '1'.
  - IPHeader.H-Length $<$ 5
  - IPHeader.TTL = 0
  - IPHeader.Checksum fails
  - AAL5Trailer.CRC32 fails

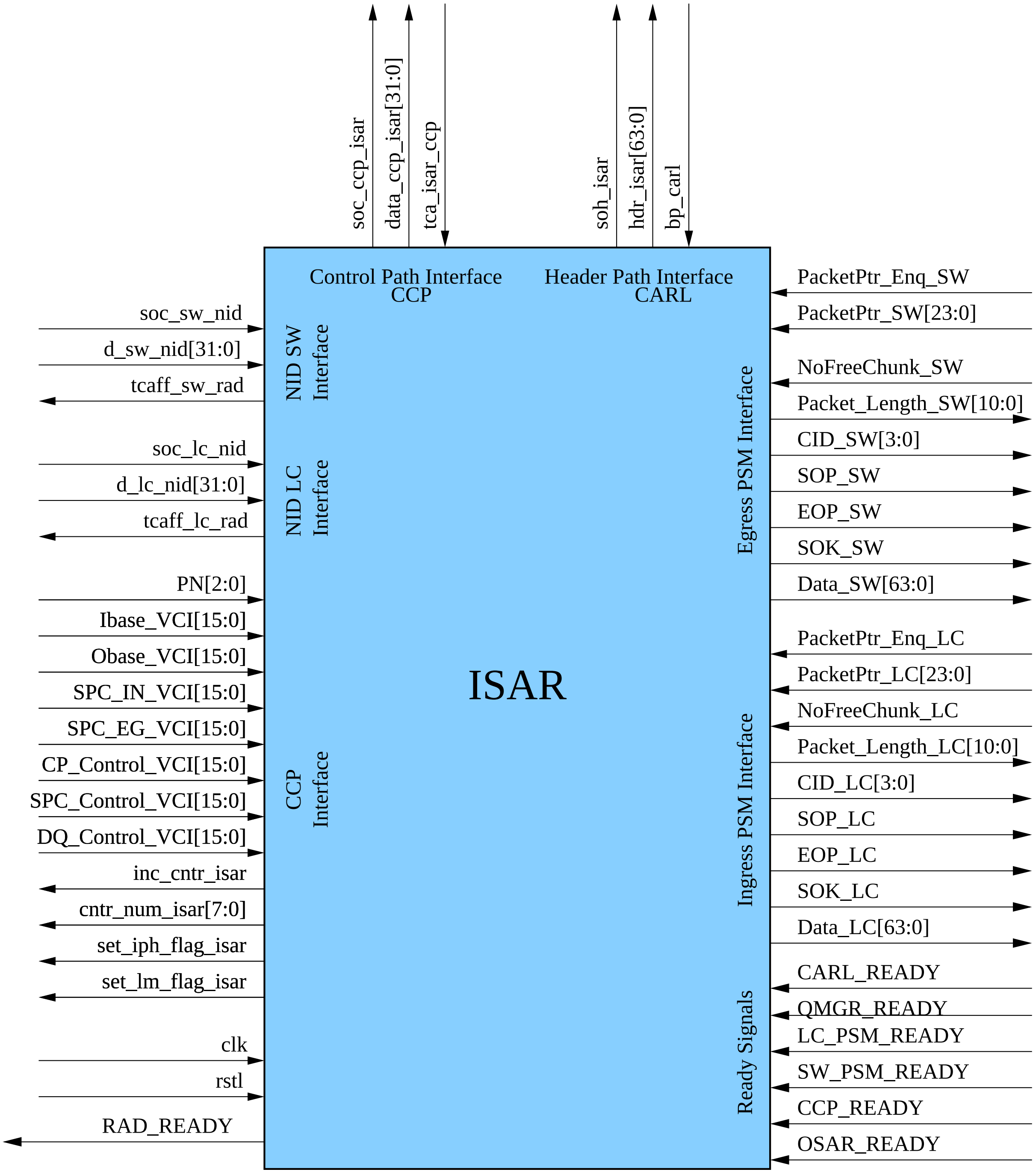


Figure 10: $isar_top$ Top-level entity of the Input Segmentation and Reassembly (ISAR).

  - CEILING(IPHeader.TotalLength/4) + ShimSize $\neq$ AAL5Trailer.Length, where ShimSize is in term of number of 32-bit words as defined below:
    - 0 for packets arriving from line card (VCI = IBase_VCI + [0:3])
    - 2 for packets arriving from switch (VCI = OBase_VCI + [0:7]).
    - 4 for packets arriving from packets arriving from SPC (VCI = SPC_IN_VCI or SPC_EG_VCI).

- **RC** is set to '1' unless any of the following condition occurs in which case it is set to '0':
  - packet coming on SPC_IN_VCI or SPC_EG_VCI has its **RC** bit set to '0' by SPC
  - **DP** bit is '1' (see condition above)
  - **EX** bit is '1' (see condition below)
- **NM** is set to '0' on all cases. This bit will be redefined by CARL. For packets returning from SPC, the value of **NM** flag is preserved.
- **EX** is set to '1' when IP Options exist in the packet header or the packet is not IPv4; '0' otherwise. For packets returning from SPC, the value of **EX** flag is preserved.
- **FM** is set to '1' if packet arrives on LC interface (i.e., VCI = IBase_VCI + [0:3] or SPC_IN_VCI); set to '0' if packet arrives on SW interface (i.e., VCI = OBase_VCI + [0:7] or SPC_EG_VCI). For packets returning from SPC, the value of **FM** flag is preserved.
- **TO** is set to '0' on most cases as this bit will be redefined by CARL. For packets returning from SPC with **RC bit** set to '0', the value of **TO** flag is preserved.

Internal Flags field is defined as follows:

- **DG** is set to '0' on all cases. This bit will be redefined by CARL.
- **SB** is set to '0' on all cases. This bit will be redefined by QMGR.
- **SR** is set to '1' if packet arrives on SPC_IN_VCI or SPC_EG_VCI, '0' otherwise.
- **IC** is set to '1' on all cases. This bit will be redefined by CARL.
- **FC** is set to '0' on all cases. This bit will be redefined by QMGR.
- **LP** is set to '0' on all cases. This bit will be redefined by CARL.
- **SC** is set to '1' whenever the entire packet fit in a single chunk, otherwise '0'.

Only when the Protocol field equals to TCP (6) or UDP (17) that the Source and Destination Port fields are defined with values from the IP Header. For other Protocol values, these two fields are set to 0.

ISAR must handle temporary extreme situation gracefully by performing a tail drop as late as possible. The following is what ISAR must do under different backpressure conditions.

- On receiving backpressure from PSM, ISAR will PAUSE (stop processing cells) until its Input FIFO is full. At that time, ISAR will start emptying Input FIFO and dropping these cells.

- On receiving backpressure from CARL, ISAR continues processing incoming cell until packet header is to be released to CARL. There are two input FIFOs (ingress and egress), they will be independently evaluated. If backpressure is still asserted then, ISAR will PAUSE until its Input FIFO is full. At that time, ISAR will start emptying the Input FIFO and dropping these cells.
- On receiving queue full from CCP_IN_FIFO, ISAR will drop incoming cells destined to CCP.

CCP maintains several statistic counters that are incrementable by ISAR. Below is a list of those counters. All these counters are listed at 4, 6, and 8.

- Per-VCI Input Packet Counters – increment when a good packet is received on the corresponding VCI.
- SPC Packet Counters: Complete Packet Counters (SPCI-IPC and SPCE-IPC) – increment Complete Packet Counter when a complete packet is received from SPC.
- Control Cell Counters: CP-ICC, SPC-ICC, and DQ-ICC – increment corresponding counters when received.
- Cell Drop Counter (IIC-DPC) – increment for cell dropped due to back pressure by either CARL or PSM or when CCP_IN_FIFO becomes full.
- Invalid Packet Counter (IVP-DPC) – increment when an errorneous packet is dropped. To distinguish which error triggers the count, two flags are defined, each associated to the error condition.
  - IP Header checksum error flag – set when AAL5 checksum pass, but IP Checksum fails.
  - Packet length mismatch flag – set when AAL5 checksum pass, IP Checksum pass, but AAL5.Length field does not match IPHeader.TotalLength, taking into account different sizes of shims prepended to the packet.

5.2 Packet Storage Manager (PSM)

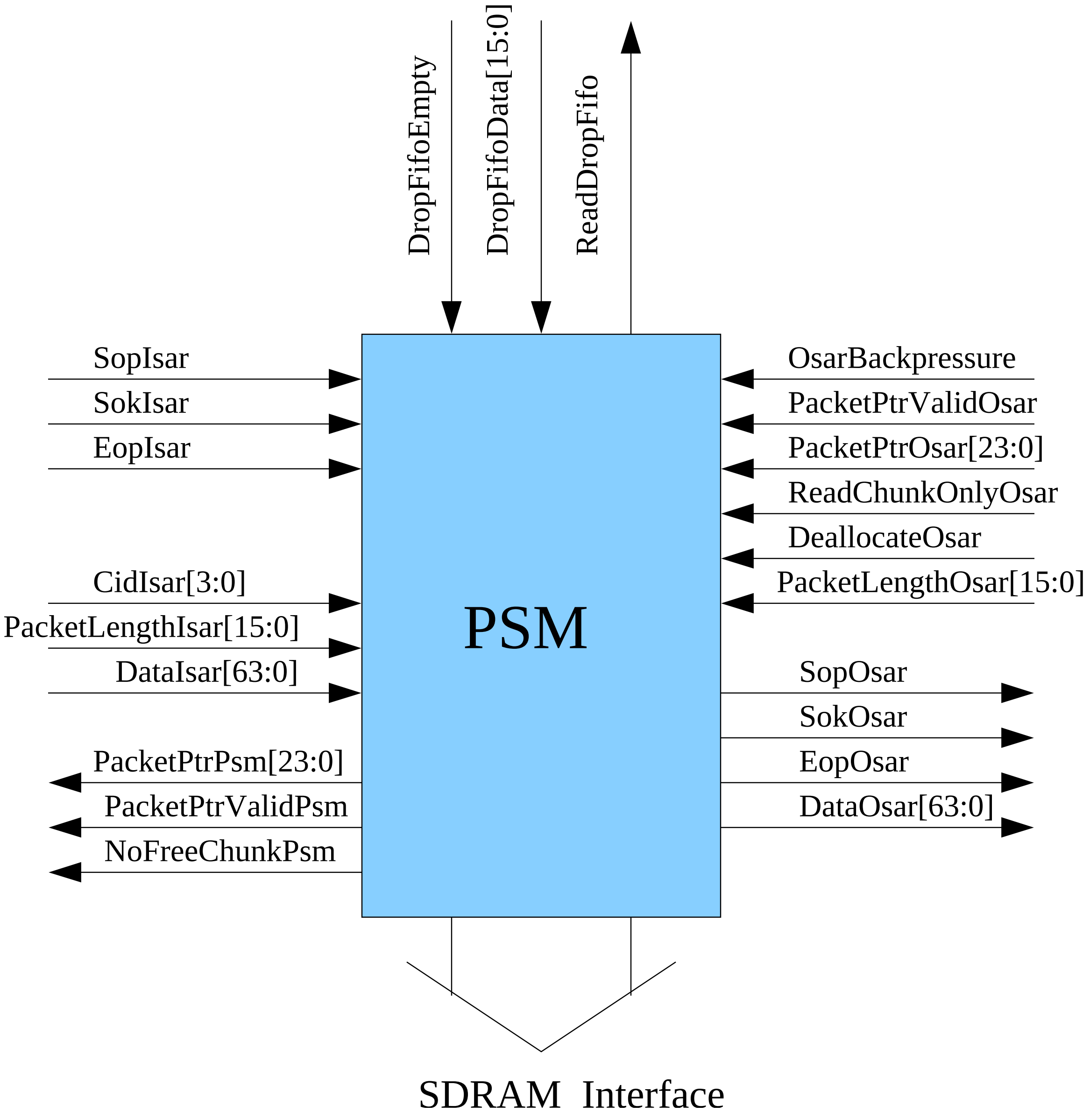


Figure 11: $PsmEntity$ Packet storage manager entity

The packet storage manager (PSM) receives the variable length packets, stores them in the off-chip SDRAM and gives them out when demanded. A variable size packet is received by PSM in the form of multiple fixed length chunks from the ISAR. The chunk size for this implementation is 128 bytes. These chunks can come interleaved for different packets. Upon reception of a chunk from the ISAR, PSM demultiplexes and appends it to the appropriate packet which is in the process of assembly. At a time, PSM can assemble 16 packets. The packet number corresponding to a particular chunk is given by the CID number associated with it. CID acts as an index to the context of the packet for which the chunk is received. (CID is the same as PID in the VHDL code of PSM).

In order to make efficient use of the buffer space, the packets are stored as the linked lists of chunks so that the internal fragmentation of the free memory space is minimized. For this purpose, the available memory is divided into chunks. When a chunk is received, it is stored in a free chunk and this chunk is appended to the linked list of the corresponding packet. A FIFO list of all the pointers to the free chunks is maintained. The free chunk pointers are pulled off from this FIFO as and when needed and allocated to the chunks received.

While a chunk is being written into the SDRAM, the last chunk of a packet might be filled partially. For this chunk only the valid words need to be written into the SDRAM. This arrangement is provided with the consideration that the worst case traffic pattern might have all the chunks partially filled, in which case the goodput of the system will degrade even if the throughput is the same, since a lot of unwanted data is being written into the SDRAM with each chunk. ISAR gives the length of the packet to PSM when it sends the first chunk of the packet to PSM. PSM keeps this length in the context of the packet and calculates the number of valid words it should expect in the incoming chunk. Only the valid words are accepted and written into the SDRAM. This allows the ISAR to stream the data back to back with arbitrary chunks lengths ranging from minimum to maximum length of the chunk. However, in the current implementation of the PSM, it can write data to SDRAM in the bursts which are multiples of 4. Thus, if the remaining number of words to be written is not a multiple of four then PSM converts it into the next larger value which is a multiple of four and writes it into the SDRAM. Note that this rounding off is done by PSM and not ISAR. However, in order to make this work, ISAR must wait for at least four clock cycles after sending a chunk with number of words not a multiple of four. This allows PSM to write this chunk properly and be ready for the next chunk. After a chunk is received, PSM fetches a new free-pointer from the free list for the next chunk to come on the same CID. It takes at least four clock cycles of latency to fetch this free-pointer from the on-chip free list. Hence the distance between any two back-to-back SOK (Start of Chunk) coming on the same CID must be at least five(?) clock cycles to allow PSM to replenish the free pointer.

To read a packet out from PSM, OSAR gives the pointer to the packet (which is a pointer to the first chunk in the packet). PSM reads the chunk starting from this pointer and retrieves the next pointer from this chunk. It then reads the next chunk from the retrieved pointer and obtains the next pointer. This is continued until PSM encounters NULL pointer as next pointer in a chunk. Just like in writer side, reader side accepts packet length from OSAR and keeps track of how many valid bytes are to be read out from the SDRAM. Depending on the number of bytes left, PSM can give the appropriate read burst length to SDRAM and just read valid number of words from SDRAM. Again, reading operation too is done in bursts which are multiples of four. Thus, it is possible that PSM fills the last chunk with some junk data to make the burst length a multiple of four. Since OSAR already has the length of the packet, it can extract the appropriate number of bytes from the last chunk that PSM reads out.

As chunks are read out, they become available for reuse. Hence, pointers to chunks are appended to the free pointer list. However, if a packet is a multi-copy packet then it has to be read out multiple times. Hence the chunks of this packets are read out, they are not free for reuse unless the packet has been read out for the last time. OSAR gives a signal with the packet pointer which tells if chunks of the packet should be freed after they are read out. If chunks should not be freed then they are not appended to the free list. Sometimes, OSAR needs to read just a specific chunk and not the entire packet. OSAR gives the packet pointer and instructs PSM to read just the first chunk of the packet. In this case, the chunk read out is not freed. It is freed only when the entire packet to which it belongs to is read out.

If a packet has to be discarded for some reason then Queue Manager gives the pointer to this packet and instructs PSM to discard the packet. PSM reads this packet out, frees the chunks occupied by it and appends free pointers to the free list. When packet is discarded, only the chunk pointers freed by the packet are of interest and not the remaining data. Hence the chunks are not read completely while a discard packet is being read but only the first word of the chunk which contains the next pointer is read out. However, since the SDRAM operates at four word boundary as mentioned above, single word read is not possible and hence first four words of the chunk are read. From the packet length received with the discard packet pointer, PSM figures out if this is a single chunk packet or a multiple chunk packet. If it is a single chunk packet then the packet need not be read at all. In this case the Packet Pointer supplied by OSAR is directly appended to the free list without reading the packet pointed to by it. These techniques help make efficient use of the memory bandwidth.

The list of free chunk pointers is maintained as a FIFO. This FIFO can be too big to fit on the on-chip RAM. Hence it is kept in SDRAM. However, if every time a pointer is retrieved or pushed into the FIFO in the SDRAM then there is a bigger latency involved in accessing the SDRAM. To avoid this latency in free pointer access, some pointers from this Free pointer FIFO in the SDRAM are cached on the on-chip block RAM and a mini free pointer FIFO is maintained on the chip. When free chunks are needed for writing incoming chunks, they are pulled off from mini-free-list. Likewise, when chunks are freed after packets are read out, free pointers are appended to the mini-free-list. When mini-free-list grows beyond a threshold (presently configured as 496), the excess pointers are pushed back to the free-list in the off-chip SDRAM. Also, when the number of free pointers in mini-free-list drops below a threshold (presently configured as 32), it is replenished by fetching the free pointers from the free-list in the off-chip SDRAM. Both, pushing and fetching pointers to and from the SDRAM list happens in the units of eight SDRAM words each containing two pointers. The buffer is declared to be full exactly when there are no free pointers in the on-chip FIFO and off-chip FIFO. At this point, PSM stops accepting data and any excess data given to PSM is dropped, including the chunk data that is already in the process of transfer. Hence, even a progressing chunk should be considered corrupt if the BufferFull indication is asserted in the middle of the transfer. (Although, PSM indicates that there are no free pointers left for data storage, actually there are free pointers left over in the PID context registers CurrentPointer and NextPointer of each PID context. PSM doesn't use those if the rest of the buffer is full) The future version of the PSM should output the number representing the free chunks left instead of a signal indicating buffer-full. This will allow ISAR to decide when exactly it should stop supplying data to the PSM. This might allow more flexible buffer management too.

5.3 Classification and Route Lookup (CARL)

Classification and Route Lookup (CARL) determines the processing and queuing actions performed for each packet based on the packet header fields received from ISAR. As shown in Figure 12, CARL employs three distinct classification engines:

- *Route Lookup* performs a Longest Prefix Match (LPM) on the IPv4 destination address of packets arriving from the link (LC) only
- *General Filter Match* performs an exhaustive search on a small set of general filters specifying a Longest Prefix Match (LPM) on the source and destination IP addresses, range matches on the source and destination port numbers, and exact match on the protocol field; general filters have local priority and may be *exclusive* or *non-exclusive*; general filter match may be performed on packets arriving from the link (LC), switch (SW), both, or neither as specified by the GM_Path register
- *Exact Filter Match* performs an exact match on the packet header 5-tuple for packets arriving from both the link (LC) and switch (SW); primarily used for software configured reserved flows

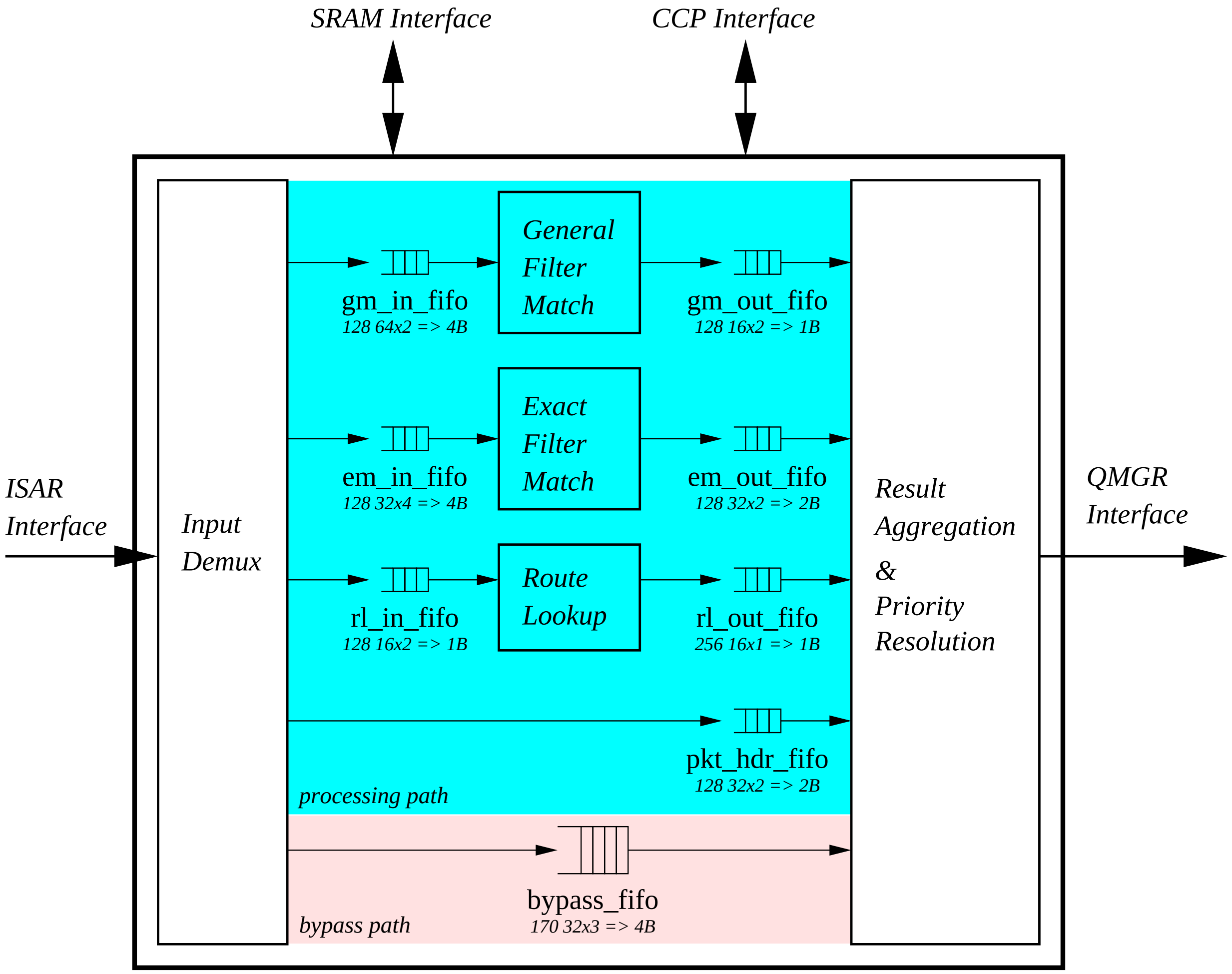


Figure 12: $carl_block$ Block diagram of the Classification and Router Lookup (CARL) block.

The Input Demux module feeds the appropriate packet header fields to each search engine. Packets requiring classification are placed on the *Processing Path*. Header fields of ingress packets arriving from the link (LC) are always sent to the Route Lookup and Exact Match engines; they are sent to the General Match Engine if the GM_Path register is set accordingly. Header fields of ingress packets arriving from the SPC (after processing) that have the reclassify flag set (RC) are treated in the same manner. Header fields of egress packets arriving from the switch (SW) are always sent to the Exact Match engine; they are sent to the General Match Engine if the GM_Path register is set accordingly. Header fields of egress packets arriving from the SPC (after processing) that have the reclassify flag set (RC) are treated in the same manner. A copy of the packet header for every packet on the *Processing Path* is placed in a packet header FIFO (pkt_hdr_fifo).

Packets returning from the SPC that do not require reclassification or exception packets (IP options, drops, etc.) are placed into the *Bypass Path*.

The Result Aggregation and Priority Resolution (RAPR) module services the *Processing Path* and *Bypass Path* in round-robin fashion, updates shim fields, and passes packet headers to the Queue Manager (QM) input FIFO. For packets on the *Processing Path*, the RAPR reads the appropriate lookup engine output FIFOs, performs priority resolution of the lookup results, updates shim fields, and makes copies of packet headers, if necessary. Packet ordering is maintained among packets on the *Processing Path* and packets on the *Bypass Path*, but not between the two paths. For example, a packet on the *Bypass Path* may leapfrog a packet or multiple packets on the *Processing Path*. This is acceptable as long as the SPC maintains the same treatment on all packets belonging to the same flow as whether or not they are to be reclassified when returned to FPX-RAD. The top level signals of the Classification and Route Lookup (CARL) module are shown in Figure 13.

In order to sustain the combined throughput of 3.2 Gb/s, CARL must process approximately 7M packets per second. Under nominal conditions, CARL can provide this level of performance. Note that there are conditions in which CARL will not sustain line speed lookups, such as:

- The system is configured to perform general matching on both ingress and egress traffic
- The distribution of exact match filters causes long linked lists in memory

The following sub-sections provide a detailed description of the operation of each sub-block.

5.3.1 Input Demultiplexor

Note that there is no FIFO between ISAR and CARL. The Input Demultiplexor receives packet header frames from the ISAR and directly copies header fields to the appropriate FIFOs. If the Re-Classify (RC) flag is not set, the Input Demultiplexor forwards the packet header frame to the *bypass path* FIFO. The frame format for packet header frames in the *bypass path* FIFO is shown in Figure 14. If the Re-Classify (RC) flag is set, the Input Demultiplexor copies packet header fields to the appropriate search engine input FIFOs. Each classification engine ensures in-order delivery of results in dedicated result FIFOs. Note that packet header frame fields not used for classification are placed in the packet header FIFO on the *processing path*. The frame format for packet header frames in the *processing path* packet header FIFO is shown in Figure 15. The frame formats for the input FIFOs of each search engine are discussed in the following sub-sections. The input frame format for the General Match engine is shown in Figure 17; the input frame format for the Exact Match engine is shown in Figure 22; the input frame format for the Route Lookup engine is shown in Figure 27.

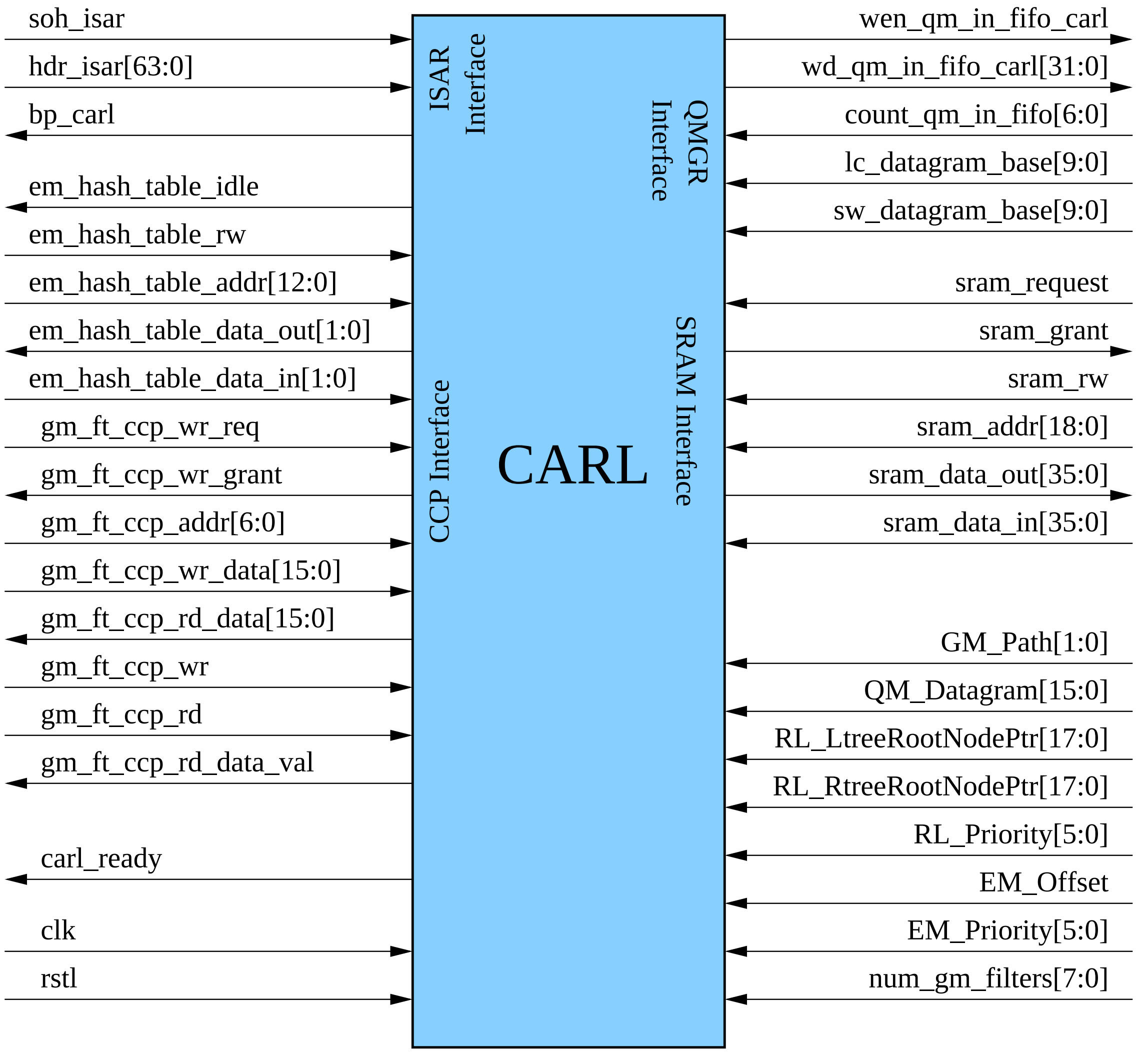


Figure 13: $carl_top$ Top-level entity of the Classification and Router Lookup (CARL) block.

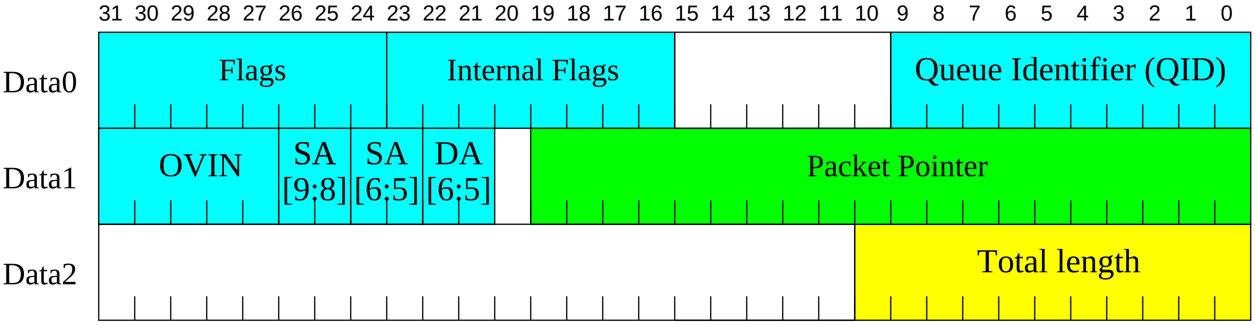


Figure 14: $carl_bypass_fifo_frame$ Format of data in Bypass FIFO.

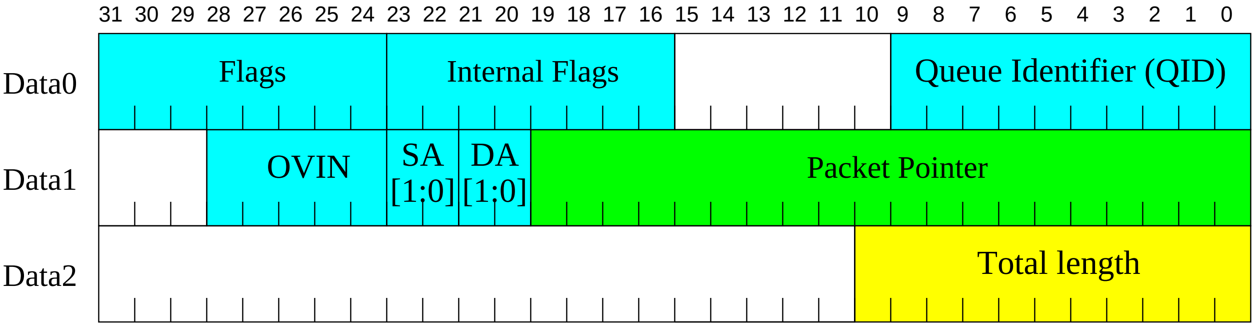


Figure 15: $carl_pkt_hdr_fifo_frame$ Format of data frames in the Packet Header FIFO (*pkt_hdr_fifo*).

5.3.2 Result Aggregation and Priority Resolution (RAPR)

The Result Aggregation and Priority Resolution (RAPR) serves packets in the *bypass path* and *processing path* in round-robin fashion. The RAPR forwards packets in the *bypass path* FIFO directly to QMGR without any processing. When serving packets in *processing path*, RAPR reads the three search engine result FIFOs as well as the packet header FIFO in parallel. Based on the lookup results and their associated priority, the RAPR decides what route, filter, or set of filters to apply to the packet. There are 64 priority levels; hence, priority is specified by a 6-bit value. 0 is the highest priority and 63 is the lowest priority. Every exclusive and non-exclusive General Match filter stores its own priority. All Route Lookup entries share the same priority which is stored in a control register. Likewise, all Exact Match entries share the same priority which is stored in a control register. See Section 5.6 for details on control register updates and default values.

A maximum of four matches will be returned for any packet: an exclusive General Match filter, a non-exclusive General Match filter, an Exact Match filter, and a Route Lookup entry. To avoid confusion, it is recommended that administrators avoid setting General Match filter priorities to the same priority level as Exact Match filters and/or Route Lookup entries. Note that non-exclusive General Match filter priority is only used to select the best matching non-exclusive filter. If a non-exclusive result is returned, then it will be applied to the packet in addition to the highest priority result from among the exclusive General Match, Exact Match, and Route Lookup results. In the case that these three results share the same priority, the exclusive General Match filter takes priority. In the case that Exact Match and Route Lookup results share the same priority and are the highest priority results, the Exact Match filter takes priority.

When a General Match filter matches (non-exclusive or highest priority exclusive match), the RAPR must perform a lookup to an on-chip *Action* table to retrieve the actions associated with the filter and update the filter counter. Thus, General Match filter counters are only incremented when the filters win the priority resolution and are applied to a packet. Currently, counters in the Route Lookup and Exact Match engines are updated when an entry is matched (prior to priority resolution).

The resulting packet header frames are transmitted to the Queue Manager according to the interface detailed in Section 6.4. For exclusive matches (from General Match, Exact Match, or Route Lookup), the **IC** (Initial Copy) and **LC** (Last Copy) bits will both be set. In the case of a non-exclusive filter match or multi-copy traffic, the actions will include making multiple copies of the header. When making the copies, the first copy to be released will have **IC** bit set, while subsequent copies will not. Likewise, the last copy to be released will have **FC** bit set, while previous copies will not. The RAPR also adjusts the **Copy_Count** field accordingly (“00” - one copy, “01” - two copies, “10” - three copies, “11” - not used).

5.3.3 General Filter Match

The primary purpose of this module is to apply filters to flows or groups of flows for packet filtering and/or monitoring. Figure 16 shows top level blocks of this module. The GM module performs an exhaustive search on a set of 32 general filters specifying a Longest Prefix Match (LPM) on the source and destination IP addresses, range matches on the source and destination port numbers, and exact match on the protocol field. (Note that the number of filters supported by the GM engine may be changed in the VHDL by altering the num_gm_filters signal to the CARL entity. Care should be taken in adjusting this value, as it affects performance.) General filters may be applied to ingress packets, egress packets, or both as specified by the **GM_Path** control register. If **GM_Path(0)** is set to 1, then ingress packets are sent to the General Filter Match module. If **GM_Path(1)** is set to 1, then egress packets are sent to the General Filter Match module. Packets returning from the SPC that need to be reclassified (RC bit is set) are forwarded to the General

Filter Match module. The load on the module depends on which paths are active. The General Filter Match module is designed to support link rate traffic and SPC traffic which is 1.2 Gb/s, or for all single ATM cell IP packet case 2.344 million packets per second. The GM filter match cannot support the full duplex rate of 3.2 Gb/s; therefore, care should be taken when using general filters on both the ingress and egress path. Figure 17 shows the format of input frames to the General Match engine stored in the input FIFO.

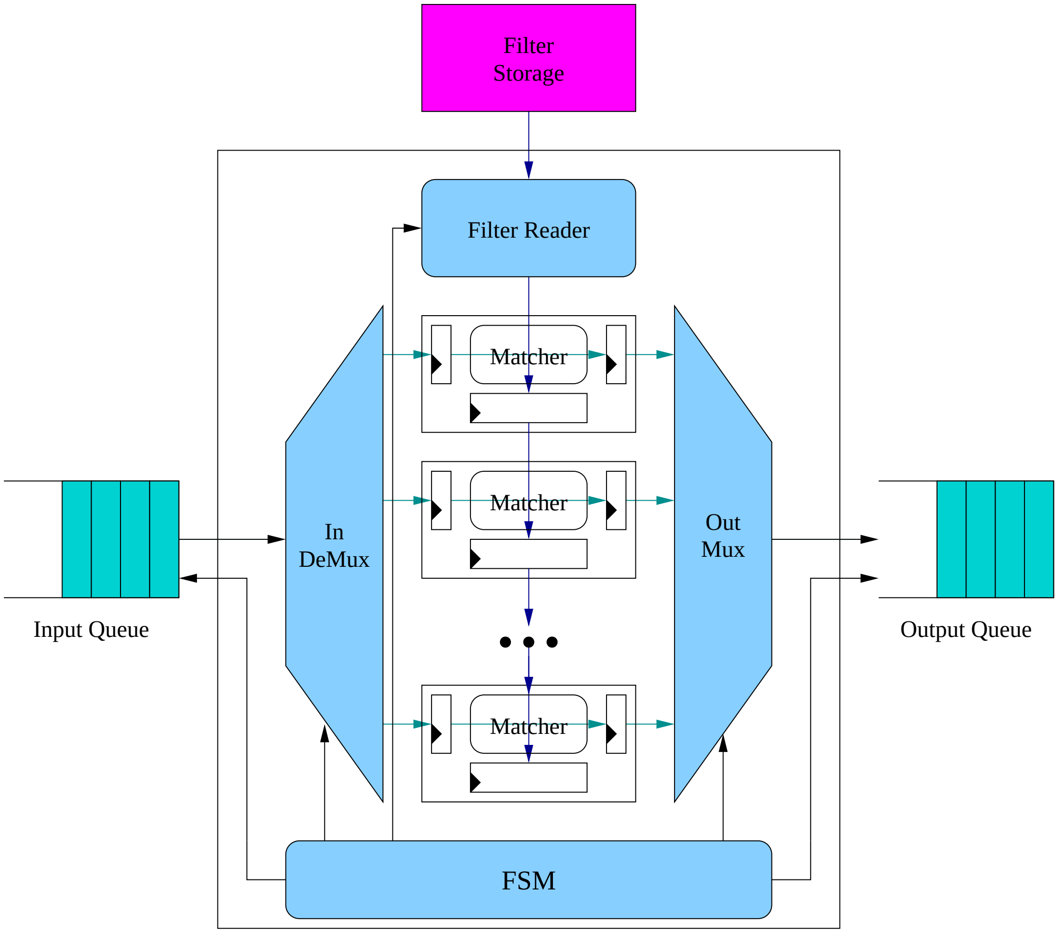


Figure 16: $carl_gm_block$ Block diagram for Generic Filter Match module.

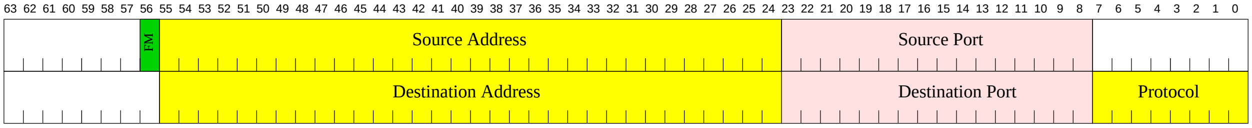


Figure 17: $carl_gm_in_frame$ Format for inputs to the General Filter Match from the input FIFO.

A general filter contains fields that fall into two groups: *key*, which contains the match conditions and priority used to determine the best matching filter, and *action*, which indicates actions to be performed on the packet if the filter matches. *Keys* are stored in a table searched by the GM module. *Actions* are stored in a separate table accessed by the *Result Aggregation and Priority Resolution* block if a GM filter is a highest priority match. The format of entries in the *Key* table are shown in Figure 18. The fields of *Key* table entries are defined below:

- Valid (V) – 1-bit
- Priority – 6-bit, indicates the filter priority
- Exclusive (E) – 1-bit, indicates if it is an exclusive filter
- Negation (N) – 1-bit, negates the sense of matching
- FroM line card (FM) – 1-bit, FM='1' indicates that this filter is for packets coming from Line Card; FM='0' for those coming from SWitch.
- Masked Source Address – 33-bit; note that we encode prefixes by using '0' followed by all '1's as a terminating sequence; this allows us to represent a prefix by using one extra bit instead of a separate 32-bit mask; for example, the prefix 101101* is stored as 1011 0101 1111 1111 1111 1111 1111 1111 1
- Masked Destination Address – 33-bit, same as masked source address
- Low and High Source Ports – 16-bit each to define range
- Low and High Destination Port – 16-bit each to define range
- WC_Protocol – 9-bit, the lower 8-bit stores the protocol while the 9th bit is used for wild card, i.e. if set, any value in packet protocol field is a match.

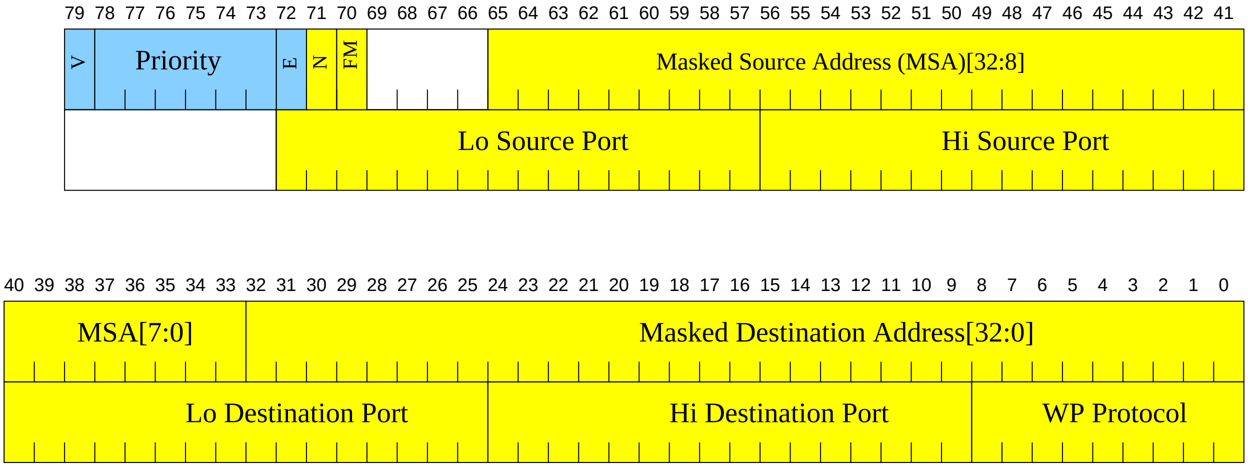


Figure 18: $carl_{g}m_{f}ilter$ Format of entries in the *Key* table in the General Filter Match module.

General filters have local priority and may be *exclusive* or *non-exclusive*; The GM module performs local priority resolution among all matching filters and returns the first highest priority exclusive filter and first highest priority non-exclusive filter. Figure 19 shows the format of the results passed to the output FIFO of the General Match module. The fields are defined as follows:

- Match (M) – 1-bit flag denoting if this word contains a matching entry
- Ex_FID – Filter Identifier for best-matching exclusive filter
- Ex_Priority – Priority level of best-matching exclusive filter

- NE_FID – Filter Identifier for best-matching non-exclusive filter
- NE_Priority – Priority level of best-matching non-exclusive filter

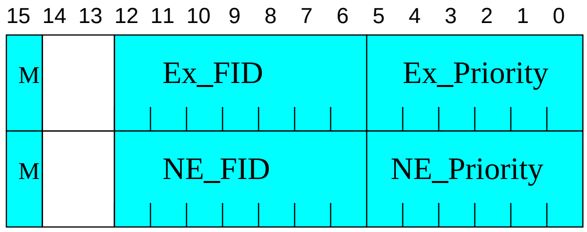


Figure 19: $carl_{g}m_{o}ut_{f}rame$ Format for outputs from the General Filter Match to the output FIFO.

If the GM module returns a non-exclusive filter or an exclusive filter with equal or higher priority than any Exact Match or Route Lookup entry matching the packet, the RAPR retrieves the *Action* information associated with the GM filter. Using the Filter Identifier (FID) as an index, the RAPR accesses the *Action* information from the *Action* table and updates the counter stored with the entry. The format of entries in the *Action* table are shown in Figure 20. The fields in the *Action* entries are defined as follows:

- Counter – 32-bit, accounts for packets that match this filter.
- TO – 1-bit, determine the direction the packet is to be forwarded to. TO = ’1’, packet will be forwarded to Line Card; ’0’, SWitch.
- Drop Packet (DP) – 1-bit, signals that the packet is to be dropped. If the packet also have a non-exclusive match that requires a copy to be made, then the copy will be dropped also.
- OVIN Valid (OV) – 1-bit, indicates that the OVIN field is a valid OVIN. When OV=’0’, Ingress packet will be assigned OVIN from Route Lookup engine if there is a match, otherwise forwarded to SPC; Egress packet will be assigned OVIN as indicated in InterPort Shim.
- QID – 11-bit, QID value assigned to packet
- OVIN – 5-bit, OVIN value assigned to packet

Note that the counter values are updated after priority resolution when the filter is applied to a packet.

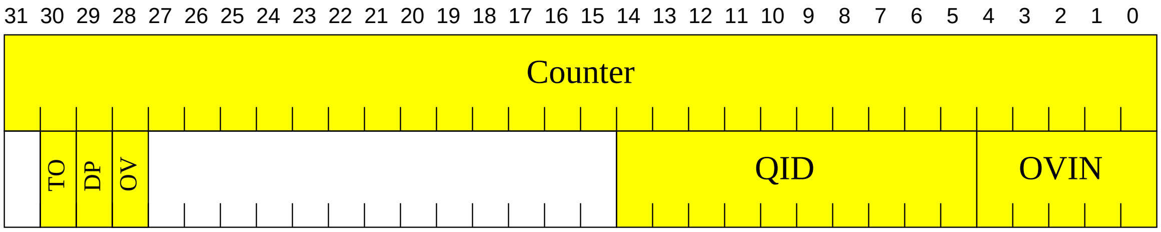


Figure 20: $carl_{r}apr_{g}m_{f}ilter$ Format of values in the *Action* table for General Match filters which is accessed by the RAPR.

5.3.4 Exact Filter Match

The Exact Filter Match block performs an exact match on the packet 5-tuple of ingress and egress packets in order to identify reserved flows. The lookup process involves a hash over the packet header fields as shown in Figure 21. Logically, the process is as follows:

- The LSBs of the source and destination IP address are concatenated to form a pointer into a static table
- If the entry at the head of the linked list matches, then the search terminates.
- If the entry at the head of the linked list does not match and there is a valid pointer to another entry in the list, then the next item in the list is retrieved.
- The search continues until a matching entry is found or the end of the list is reached.

A more detailed description of how this search is implemented is provided below.

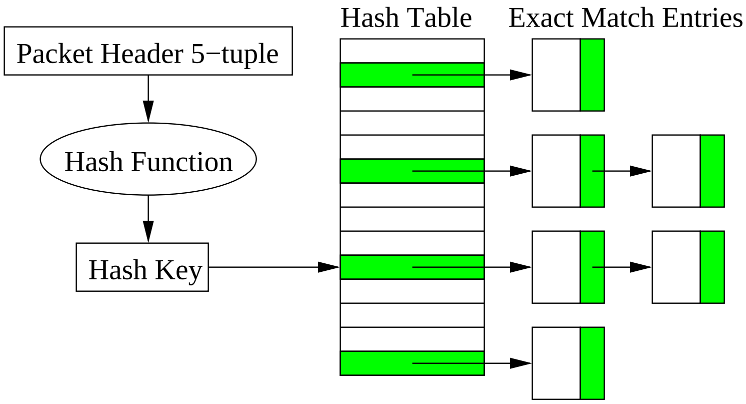


Figure 21: $carl_em_block$ Block diagram of hashing scheme for exact filter match lookups.

The format of the frames input to the Exact Filter Match block via an input FIFO are shown in Figure 22. The Exact Filter Match block must sustain the full 3Gb/s (7M lookups per second) throughput. It has 75% of the available SRAM bandwidth, as it must allocate two out of every eight clock cycles to the Route Lookup block. This provides an average of eight memory accesses per lookup.

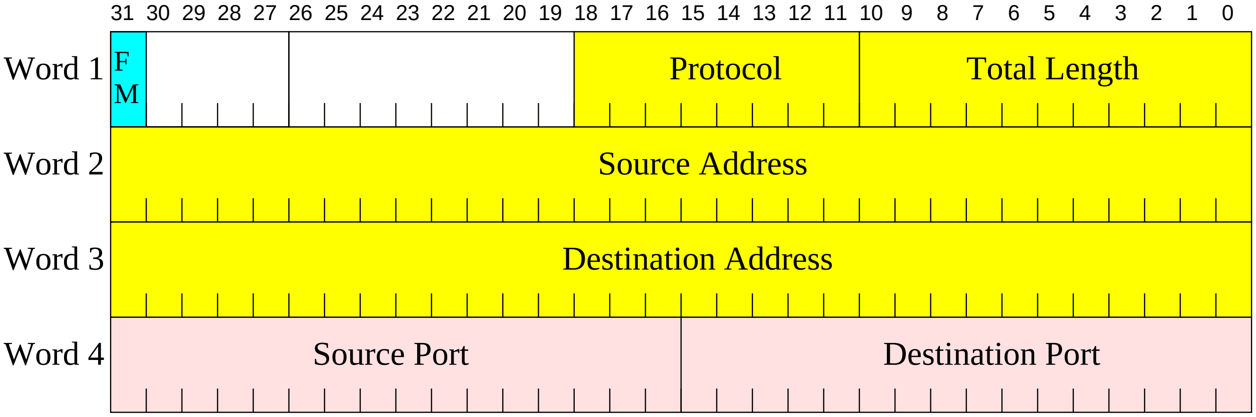


Figure 22: $carl_em_input_frame$ Frame format for inputs to the Exact Filter Match block via the input FIFO.

As shown in Figure 23, a 13-bit hash key is composed of the lower 7-bits of the IPv4 source address concatenated with the lower 6-bits of the IPv4 destination address. This key addresses an on-chip 2 x 8192 hash table requiring 4 BlockRAMs. Hash table entries contain Ingress Valid and Egress Valid bits depending on what kind of filter(s) exists in memory that match the hash key. The hash result is considered null if the valid bit corresponding to the packet direction as determined by the FM flag (0=SW/egress, 1=LC/ingress) is set to zero. If the result is not null, the 18-bit SRAM address of the head of the linked list is created by prepending a 1-bit offset (set to '1'), the FM shim flag, and the product of the 13-bit hash key multiplied by six (6). Note that the hash key may be viewed as an entry index. Since EM entries occupy six words of memory, the key must be multiplied by six and concatenated to the offset and From bits in order to generate the correct address.

A memory map of the SRAM shared between the Route Lookup and Exact Match engines is shown in Figure 24. Note that the lower half of memory is dedicated to the Exact Match table. The EM space dedicates half of its space (one quarter of the total) to ingress filter entries, and half of its space to egress filter entries. Both the ingress and egress space are divided into two buffers for head entries and list entries. Note that the size and boundaries of the buffers facilitate indexed addressing. Hash keys and pointers should be considered as indexes into the buffers. Addresses are generated by multiplying the indexes by six (6) and prepending the correct offset bits. Also note that two head entries are allocated for each hash key in each of the head entry buffer pools. This was done to simplify table management. The total capacity of the Exact Match filter table is 20,480 filters.

As shown in Figure 25, entries contain the remaining bits of the 5-tuple, flags, two OutputVINs, two QIDs, byte and packet counters, and a linked-list pointer requiring six 36-bit SRAM words. Note that the most significant bit of the second word is the linked-list pointer V (Valid) bit. This bit is set to 1 if the linked list pointer is valid; it is set to 0 if not, signaling the end of the list has been reached.

The flags are defined as follows:

- **Filter Type**: 000 = unicast software reserved flow; 001 = multicast software reserved flow
- **TO (To LC/SW)**: This flag allows components to distinguish the destination of the packet via a single bit compare. 0 = To Switch, 1 = To Line Card. Note that the flag defaults to 0 (SW) for multicast filters.
- **V0 (Valid branch 0)** When the filter type is multicast and the zero (0) branch of the tree is valid, this flag is set to 1, otherwise 0. OutputVIN0 and QID0 are used to forward the packet.
- **V1 (Valid branch 1)** When the filter type is multicast and the one (1) branch of the tree is valid, this flag is set to 1, otherwise 0. OutputVIN1 and QID1 are used to forward the packet.
- **SU (Suppress Upstream-copy)** When the filter type is multicast and the OutputVIN matches the InputVIN, an upstream copy of the packet will be generated. This flag may be set to 1 to prevent the upstream copy from being sent. The flag defaults to 0, allowing upstream copies to be sent.

Note that OutputVIN1 and QID1 will only be valid when the filter type is a multicast software reserved flow (001).

Note that multicast is a partially implemented feature and should not be relied on without consulting the designers.

Note that the bottom half of Figure 23 illustrates the mechanism for generating the physical address of exact match list entries (or entries which are not at the head of the linked list). If the packet 5-tuple matches the 5-tuple stored in the filter entry, then a matching filter is found. If the filter does not match and the next pointer valid bit is not set, then the search terminates. If the filter does not match and the next pointer valid bit is set, then the address of the next entry in the linked list is constructed as follows. The EM offset bit (set

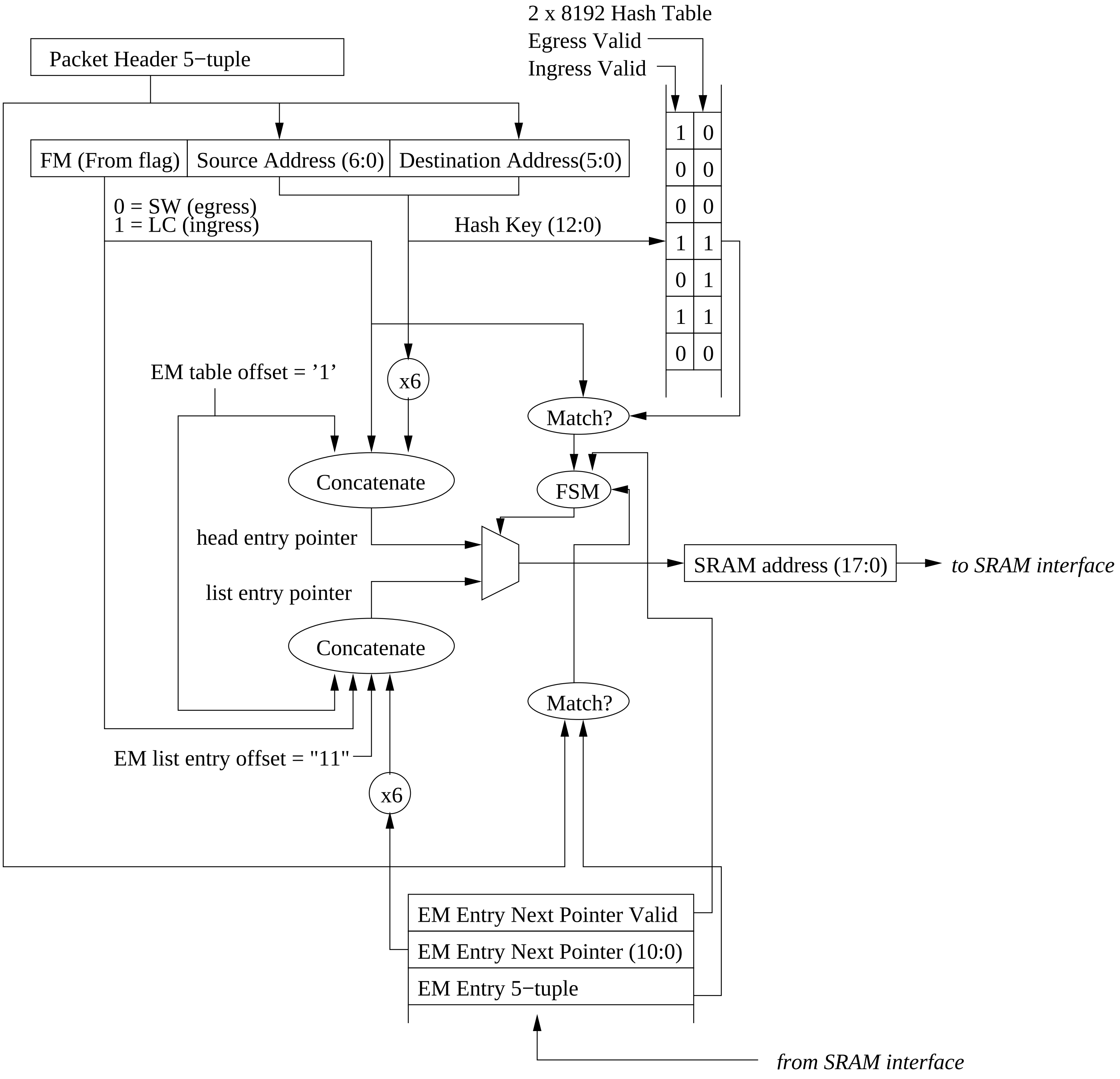
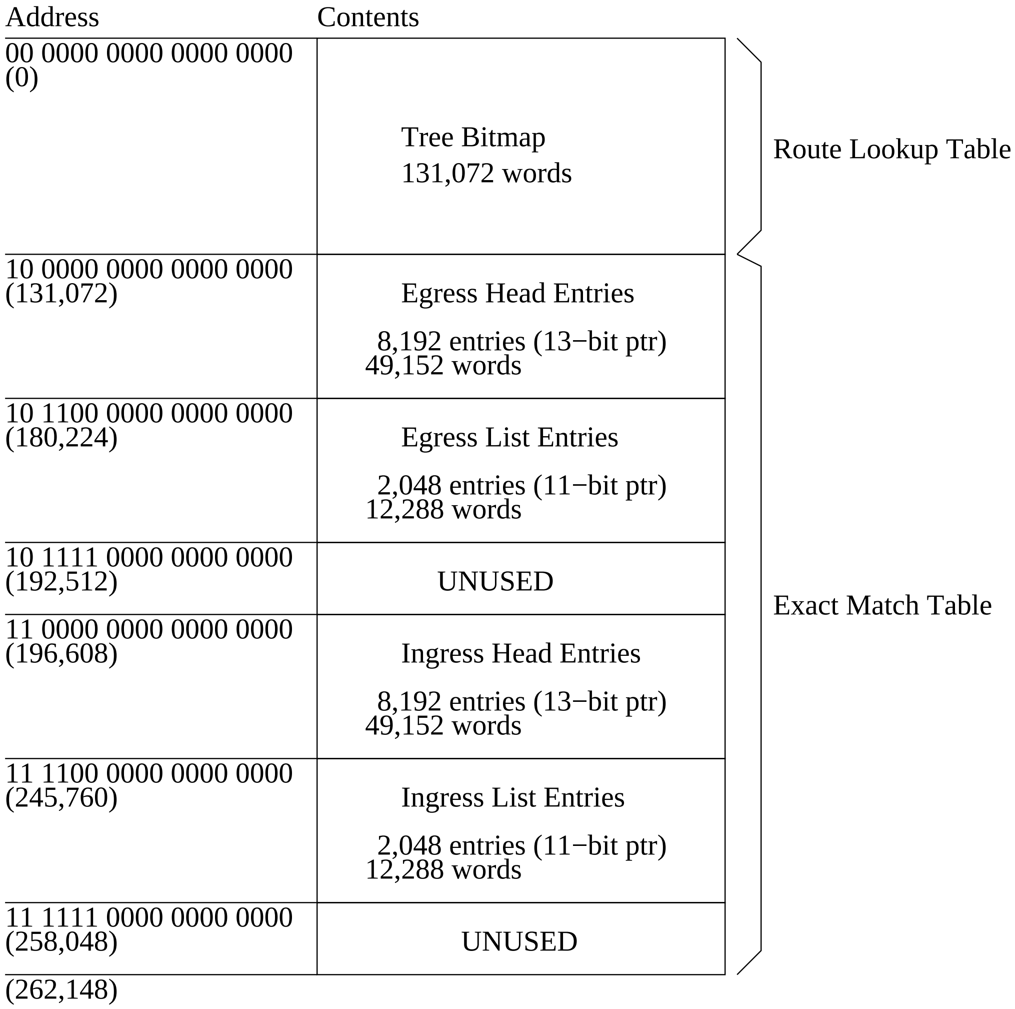


Figure 23: $carl_em_block_detail$ Detailed block diagram of hashing scheme for exact filter match lookups.



Address	Contents
00 0000 0000 0000 0000 (0)	Tree Bitmap 131,072 words
10 0000 0000 0000 0000 (131,072)	Egress Head Entries 8,192 entries (13−bit ptr) 49,152 words
10 1100 0000 0000 0000 (180,224)	Egress List Entries 2,048 entries (11−bit ptr) 12,288 words
10 1111 0000 0000 0000 (192,512)	UNUSED
11 0000 0000 0000 0000 (196,608)	Ingress Head Entries 8,192 entries (13−bit ptr) 49,152 words
11 1100 0000 0000 0000 (245,760)	Ingress List Entries 2,048 entries (11−bit ptr) 12,288 words
11 1111 0000 0000 0000 (258,048)	UNUSED
(262,148)	

Figure 24: $em_fipl_mem_map$ Memory map of the 8Mb (262,148 word) SRAM shared between the Route Lookup and Exact Match search engines.

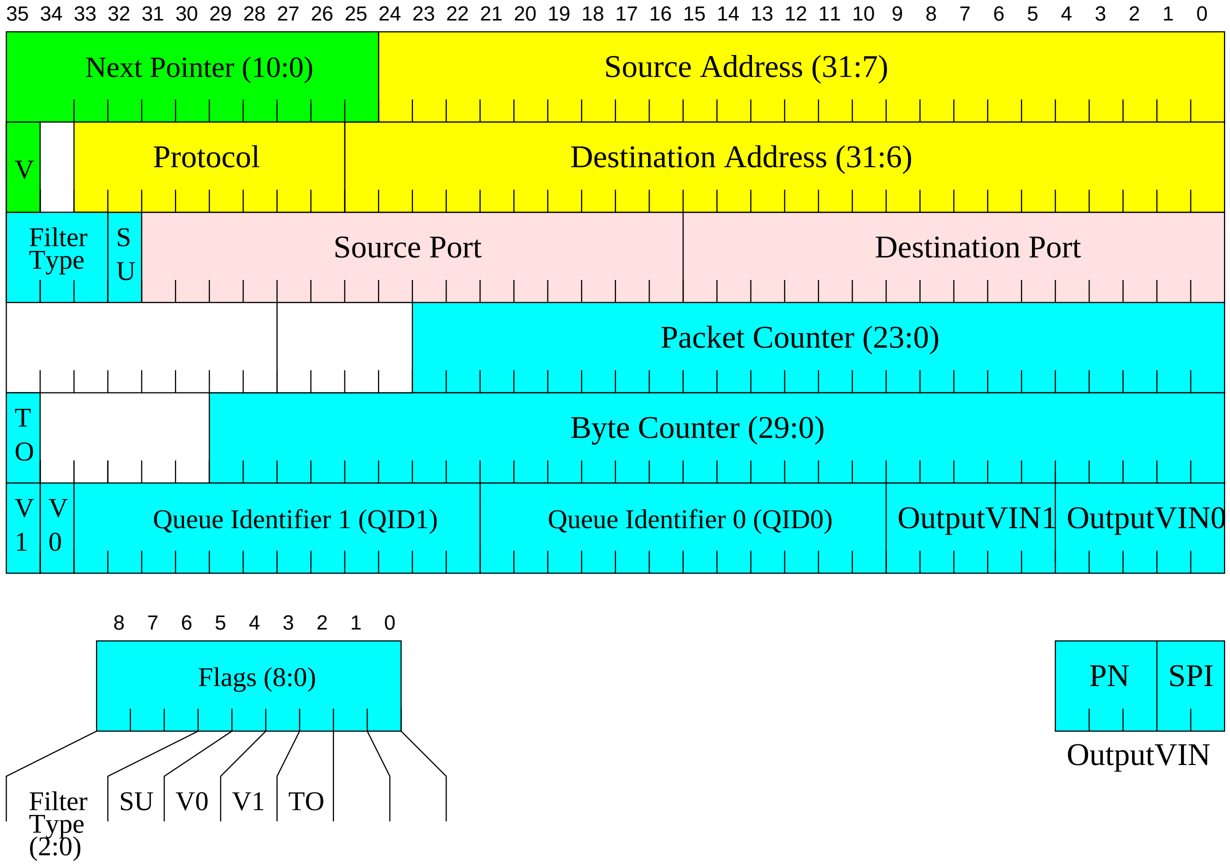


Figure 25: $carl_e m_e ntry$ Diagram of exact filter match entries.

to ‘1’), the FM flag, and EM list entry offset (set to “11”) are prepended to the product of the 11-bit next pointer multiplied by six (6). This will index the correct entry in the correct list entry buffer.

Upon finding a matching filter, the packet Total Length field is added to the Byte Counter and the Packet Counter is incremented. Both fields are written back to memory. The format of the results passed to the Exact Filter Match output FIFO is shown in Figure 26. Note that the most significant bit of Word 1 (labeled M) is set to 1 if a matching filter was found; it is set to 0 if no matching filter was found. Note that both result words will be written to the output FIFO regardless of whether or not a matching filter was found. Allowing for counter writebacks, retrieving a matching entry requires eight clock cycles requiring an average of one hash probe per lookup. Note that searches terminate when no matching filter is found and the next-pointer valid bit of the last filter searched is zero.

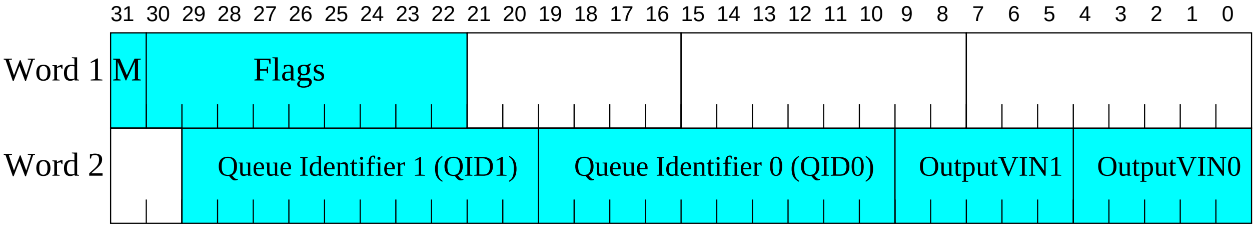


Figure 26: $carl_{e}m_{o}utput_{f}rame$ Frame format for outputs from the Exact Filter Match block to the output FIFO.

5.3.5 Route Lookup

The route lookup engine performs a Longest Prefix Match (LPM) using a compressed trie datastructure called Tree Bitmap [10]. Since this function must only be performed on traffic arriving from the link, the Route Lookup engine must sustain a throughput of 1 Gb/s or 2.36M lookups per second. Note that in this application the LPM is performed over 32 bits, the IPv4 destination address. Addresses are fed to the route lookup engine via an input FIFO in the format shown in Figure 27.

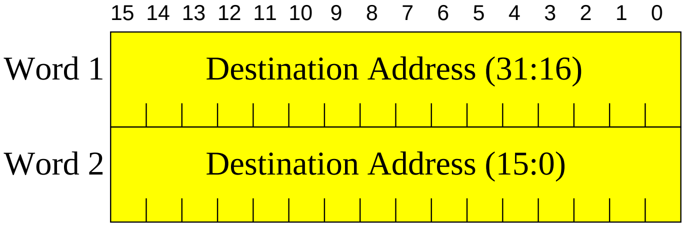


Figure 27: $carl_{f}ipl_{i}nput_{f}rame$ Format of input FIFO data for the route lookup engine.

The datastructure is searched using pipelined Fast IP Lookup (FIPL) engines as described in [11]. Note that we employ the “Split-Tree” version of the datastructre which aligns the multi-bit nodes on bit-boundaries which are a multiple of 4. This improves average lookup performance due to the dominance of 16- and 24-bit prefixes in routing tables. The first bit of the IP Destination address determines whether the search continues with the right (1) or left (0) subtree. Pointers to the root nodes of these subtrees are stored in registers that are read/write-able via the CCP. Employing a 4-bit stride, creates Tree Bitmap subtrees with a maximum depth of 8 nodes. The Tree Bitmap node format is shown in Figure 28. In this case, a worst-case lookup requires 11 memory accesses using an optimization of the level 8 nodes. With a clock speed of 75 MHz, a signal FIPL engine can achieve a worst case lookup rate of 852k lookups per second. Note that actual performance is highly dependent upon the average depth of prefixes in the datastructure. As reported in [11], performance studies using the Mae West routing database showed that employing two FIPL engines achieved a throughput of 2.95M lookups per second.

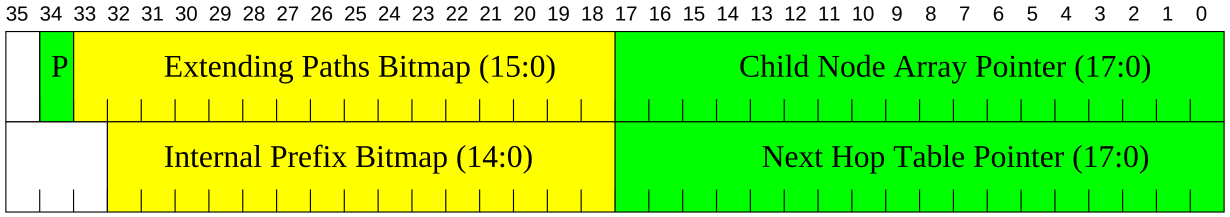

35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
P
Extending Paths Bitmap (15:0)
Child Node Array Pointer (17:0)
Internal Prefix Bitmap (14:0)
Next Hop Table Pointer (17:0)

Figure 28: $carl_rl_node$ Tree Bitmap node format.

All lookup results have the same fixed 6-bit priority, configurable via a register in the CCP. Each entry returns and OutputVIN and packet counter as shown in Figure 29. Packet counters are incremented when a matching route lookup entry is applied to a packet and written back to memory. The OutputVIN and match flag are written to the output FIFO as shown in Figure 30. The match flag is set if a route was found in the table.

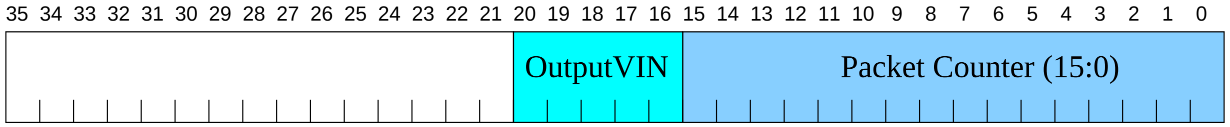


Figure 29: $carl_rl_entry$ Format of route lookup entries.

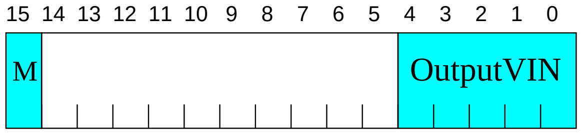


Figure 30: $carl_fipl_output_frame$ Format of route lookup results written to the output FIFO.

5.4 Queue Manager (QMGR)

Top level signals of the Queue Manager (QMGR) module are shown in Figure 31. The Queue Manager (QMGR) manages packets going to the switch fabric (SW), the line card (LC) and the Smart Port Card (SPC). When the QMGR receives a packet header from the CARL, it places the packet onto one of the queues based on the destination and Queue Identifier (QID) of the packet.

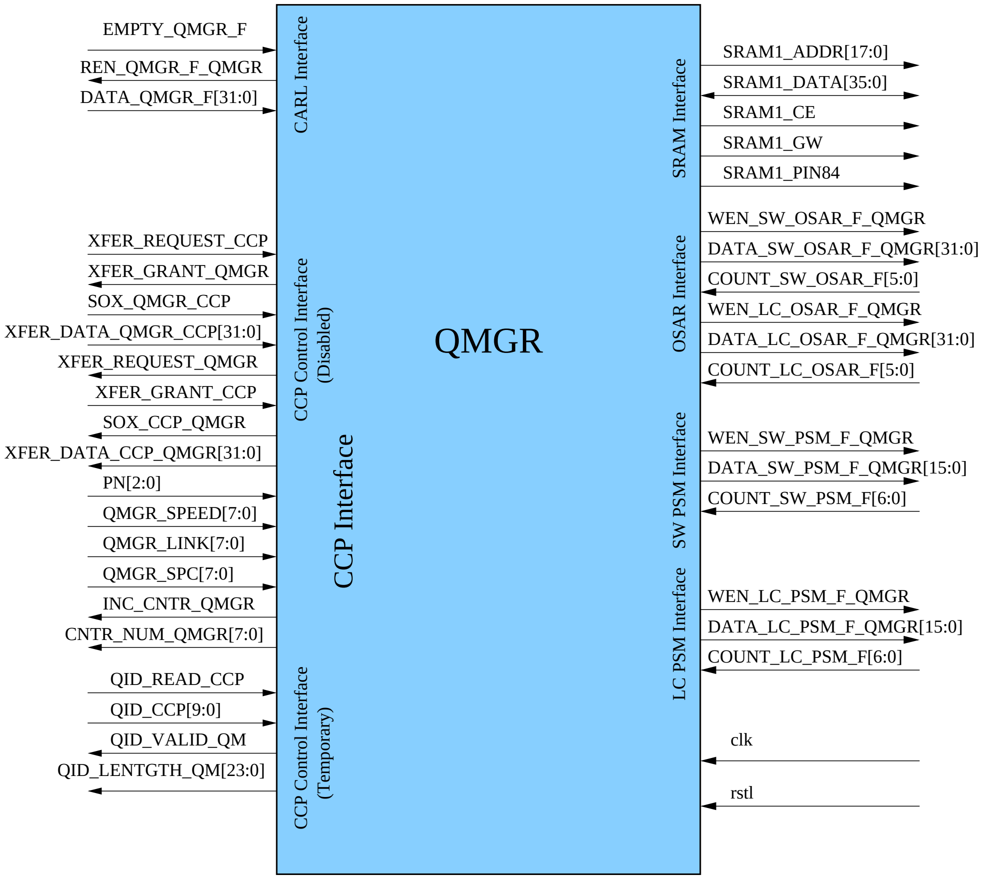


Figure 31: $qmgr_top$ Top-level entity of the Queue Manager (QMGR)

The interface between the CARL and the QMGR is shown in Figure 58.

The QMGR exams the DP (Drop Packet) flag fist. If DP = 1, the packet is to be dropped. The QMGR will put the packet into the Drop Packet FIFO between the QMGR and the PSM (Packet Storage Manager).

If the packet is not to be dropped, The QMGR determines the destination of the packets based on the flags and the QID of the packet. The TO (To LC/SW) flag indicates the destination of the packet. The packet is destined for the switch if TO = 0 and is destined for the line card if TO = 1. If the QID is within a certain range (1:127) and the SR (SPC-return) flag is not set (SR = 0), the packet should be sent to SPC for processing. The QMGR will place the packet onto one of the SPC queues. The QMGR will set the SB (SPC-bound) flag to 1. This flag is used by the OSAR to determine the destination of the packet. If the QID is within the SPC range and the SR flag is set (SR = 1), the packet has already be processed by SPC and is ready to be transmitted to the switch or the line card. In this case, the destination of the packet is determined by the TO flag described above.

The CARL (Classification and Route Lookup) can produce up to three copies of the packet. The IC

(Initial Copy) flag is set to 1 by the CARL for the first copy of the packet. The Copy Count (CpyCnt) field defines the total number of the additional copies. CpyCnt = 0 means that there is only a single copy of the packet; CpyCnt = 1 means that there is one extra copy of the packet (total of two copies); CpyCnt = 2 means that there are two extra copies of the packet (total of three copies). Multiple copies of the packet headers are queued separately based on each copy's destination and QID. However, a Common Reference entry is shared by all copies of the same packet in the QMGR. The Common Reference entry contains the copy count reference and the packet pointer to the SDRAM where the packet is physically stored. A new Common Reference entry is assigned when the QMGR receives the first copy of a multiple copy packet request from the CARL. When any one of the multi-copy packets is selected for departure, the common reference entry is visited and the copy count in the entry is updated. If the common copy count reference is zero before updating, the FC (Final Copy) flag is set to 1. In this case, the OSAR will instruct the PSM (Packet Storage Manager) to free up the chunks after reading the packet out of the SDRAM. If the common copy count reference is not zero before updating, the QMGR decrements the copy count by 1 and sets the FC flag to 0. When the PSM retrieves the copy of the packet in this case, it does not free up the chunk space occupied by the packet. For single copy packets, the FC flag is always set to 1.

The QMGR interfaces with the external SRAM where all packet headers are stored. The SRAM is partitioned into a set of two-word entries. With an 1 MB SRAM configuration, the QMGR can support up to 2^{17} single copy packets. Multi copy packets need an extra entry to store the common reference entry. The memory word format of a single copy packet header is shown in Figure 32. The Flag, the Internal Flag,

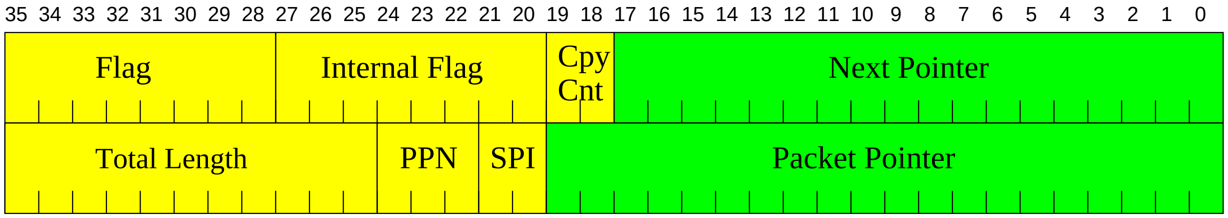


Figure 32: $sram_{s}ingle_{c}py_{p}kt_{h}eader$ SRAM Single Copy Packet Header Entry

the Total Length and the Sub Port Identifier (SPI) are taken from the packet header fields passed from the CARL. The CpyCnt field should be always 0 in this case. These two bits are also used to distinguish the single copy packet header format from the multi copy packet header format. The Next Pointer field is used to link packets in the queue. The Output VIN and the QID are not stored in the SRAM with the packets. These are queue specific fields and can be recovered from the queue the packet is placed on.

Each copy of the packet header of an multi copy packet is queued independently. The entry used to store multi-copy packets is slightly different from the one used by the single copy packets. The SRAM entry of a multi copy packet is shown in Figure 33. The Common Reference Pointer points to the Common Reference

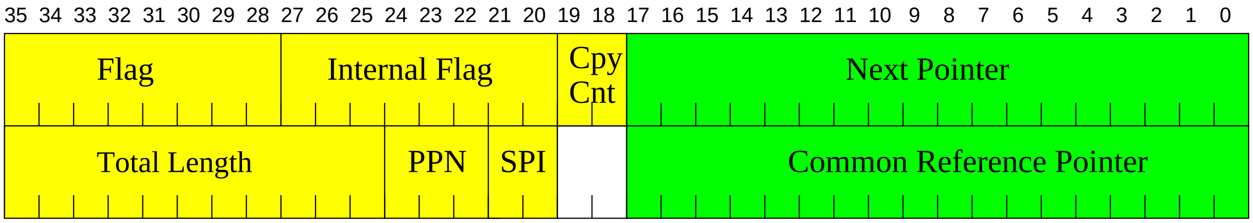


Figure 33: $sram_{m}ulti_{c}py_{p}kt_{h}eader$ SRAM Multi-Copy Packet Header Entry

entry where the copy count and the packet pointer are stored. The Common Reference Pointer Entry is shown in Figure 34.

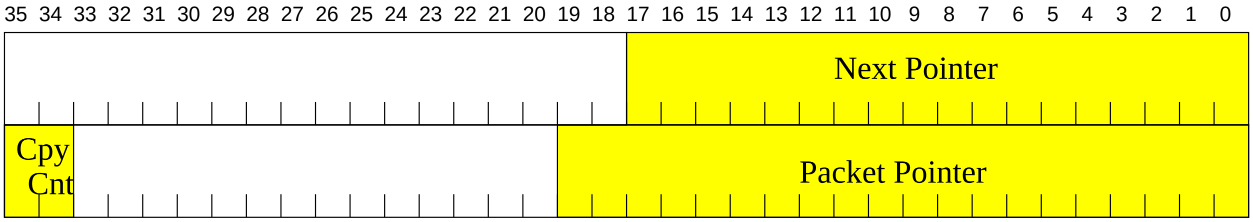


Figure 34: $sram_common_ref$ SRAM Multi Copy Packet Common Reference Entry

The QMGR supports 512 queues. Queues are identified by the Queue Identifiers (QID). It supports 127 queues destined for the SPC, 127 queues coming back from the SPC, 8 ingress datagram queues, 64 egress datagram queues, and 184 other reserved flow queues shared between ingress and egress. The QID assignments are as follows:

- 1 - 127: To SPC (1-7 are FPX reserved SPC queues. Software can only use 8-127)
- 128 - 255: From SPC (Internal to the QMGR, not software configurable.)
- 256 - 439: Reserved flow queues
- 440 - 503: Line card (egress) datagram queues
- 504 - 511: Switch (ingress) Virtual Output Qeues (VOQs)

Packets destined for the line card or SPC are queued based their QIDs. QIDs for packets going to the SPC are in the range of 1 to 127. An offset of 128 is added to the packets coming back from the SPC. The new QID used in the QMGR is determined by QID + SPC_QID_BASE, where SPC_QID_BASE = 128. The SPC return packets are then placed in either the switch side queues or the line card side queues.

The egress queue manager rate limits the traffic it sends to the SPC and line card using two token bucket regulators. By default the SPC is rate limited to 200Mb/s and the line card to 600Mbps, although this may be altered by writing to registers 0x0e (signal QM_SPC) and 0x0d (signal QM_LINK). These registers set the bucket's token fill rate which in turn sets the average sending rate. The actual sending rate depends on the FPX's clock rate F and the token fill rate z according to R(Kbps) = F*z/1024. The current clock rate is 62.5MHz so the sending rate is approximated by R(z) = z*61(Kb/s).

In addition to rate limiting the sending rates, the egress queue manager assigns both a threshold and weight to each queue. The threshold is used to determine when packets are to be dropped. If adding a packet to a queue would cause the length to exceed the threshold then the packet is dropped. The weights (i.e. quantums) are used to implement a weighted deficit round roubin (WDRR) service disclipline across all ready queues.

The ingress queue manager implements per VOQ rate control using eight token bucket regulators. A threshold and token rate parameter is associated with each of the eight VOQs. As with the egress side, the threshold implements the drop policy and token rate sets the average sending rate to the associated output port.

5.5 Output Segmentation and Reassembly (OSAR)

Top level signals of the Onput Segmentation and Reassembly (OSAR) module is shown in Figure 35.

The OSAR module receives packet retrieval requests from the Queue Manager (QMGR). It forwards the request to either the Ingress Packet Storage Manager (Ingress PSM) or the Egress Packet Storage Manager (Egress PSM) based on IE field in shim flag. Packets retrieved by the PSM contain one or more 64-bytes chunks. The OSAR formats data into AAL5 frames and sends the ATM cells to either the line card (LC) or the switch (SW) interfaces based on the destination. Control cells received from the Control Cell Processor (CCP) are multiplexed onto SW interface as well.

The OSAR interfaces with two independent PSMs (LC and SW PSM). The packets originally coming from the LC interface are stored in the LC PSM and the packets coming from the SW interface are stored in the SW PSM. There are two separate FIFOs between the QMGR and the OSAR. Packets retrieval requests for packets stored in the LC PSM are in the LC OSAR FIFO, and the packets requests for packets stored in the SW PSM are in the SW OSAR FIFO. The OSAR will forward the requests to the correct PSM interface. After the packets are retrieved from the PSM, the OSAR needs to switch the packets to the correct NID/RAD interfaces at AAL5 level. In other words, two packets with the same destination interface (LC/SW) retrieved from the LC PSM and the SW PSM respectively can not be interleaved. For a specific NID/RAD interface (LC/SW), the OSAR must transmit the entire AAL5 frame received from a single PSM (LC/SW) before it can transmit an AAL5 frame received from the other PSM (SW/LC). This is necessary to avoid interleaving AAL5 frames with the same VCI.

The OSAR checks the LC (Last Copy) flag in the packet header. If the LC flag is set, the OSAR generates LST_CPY_OSAR signal to the PSM. In this case, the PSM frees the chunk pointers after retrieving the packet. The Total Length field is also sent to the PSM.

For packets going to the SW, the OSAR needs to perform the following operations before transmitting the packet to SW interface. The OSAR inserts Flags, Input VIN and Output VIN in the InterPort Shim. The OSAR segments the packet into ATM cells, and creates an AAL5 frame for each packet. The VCI of all cells in the AAL5 frame is determined by the Output VIN (VCI = OBased_VCI + Output VIN PN). AAL5 padding is set to zeros. UU and CPI fields in AAL5 trailer are all zeros. Length field in AAL5 trailer includes shim fields and the length of the packet (excluding AAL5 padding and AAL5 trailer). AAL5 checksum is computed over the payloads of all ATM cells in the frame (excluding checksum field).

For packets going to the LC, the OSAR removes the shim and updates the IPv4 checksum using the incremental update algorithm. The IPv4 TTL is decremented. The OSAR updates the IPv4 header checksum using the incremental update algorithm. The OSAR segments the packet into ATM cells, and creates an AAL5 frame for each packet. The VCI of all cells in the AAL5 frame is determined by the Output VIN (VCI = IBased_VCI + SPI). AAL5 padding is set to zeros. UU and CPI fields in AAL5 trailer are all zeros. Length field in AAL5 trailer excludes AAL5 padding and AAL5 trailer. AAL5 checksum is computed over the payloads of all ATM cells in the frame (excluding checksum field).

For packets going to SPC for processing (SB = 1), the OSAR sends the packet out differently depending on where the packet came from. If the packet comes from the LC, the packet should be transmitted on the RAD_LC port, with the VCI set to SPC_IN_VCI. If the packet comes from the SW, the packet is transmitted on the RAD_SW port, with the VCI set to SPC_EG_VCI. The OSAR inserts Flags, Input VIN, Output VIN, QID, Total Chunks and Queue Length fields in the IntraPort Shim.

The OSAR retrieves the entire packet from the PSM, inserts IntraPort Shim, segments the packet into ATM cells, creates an AAL5 frame for the packet and sends it to the SPC.

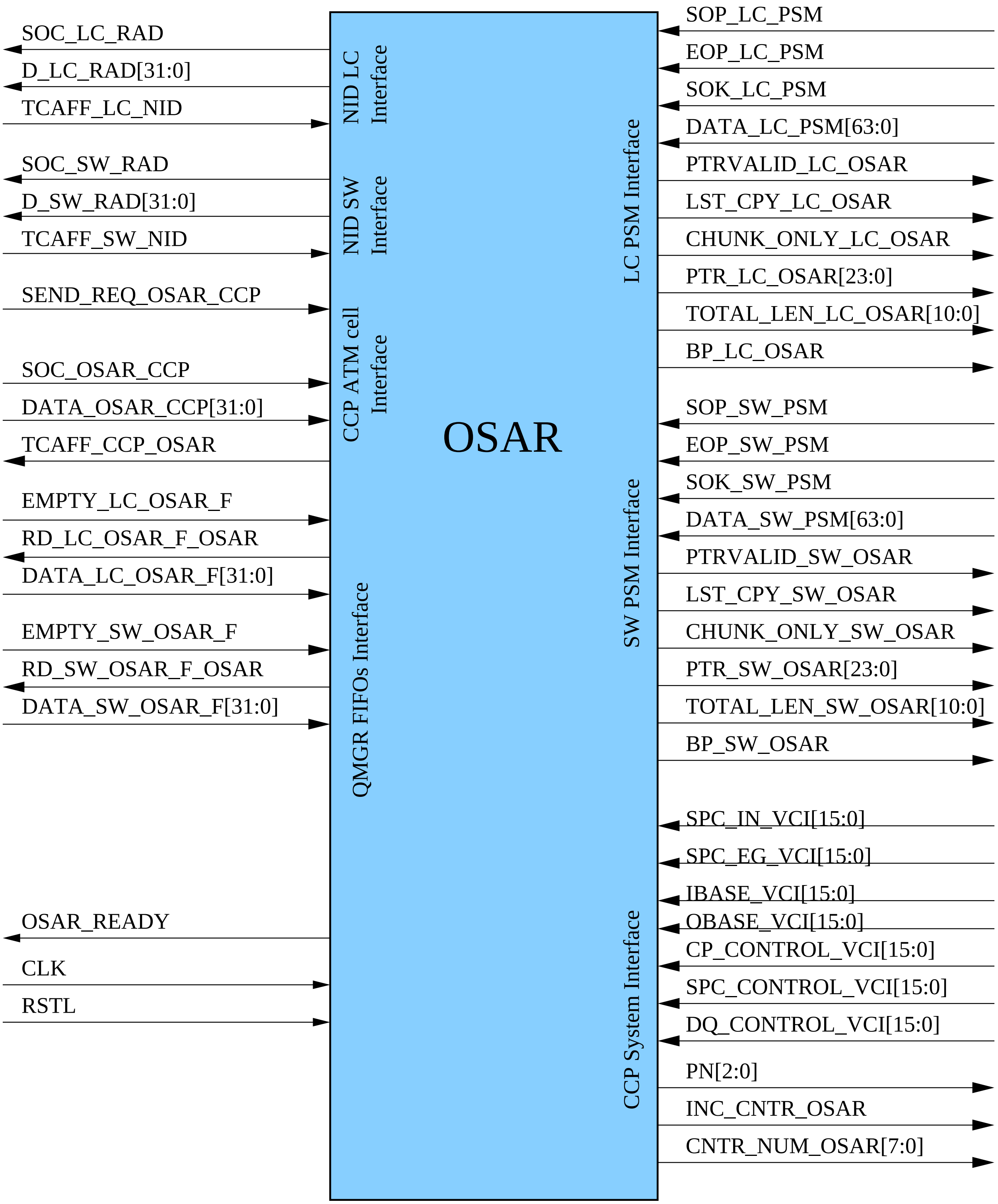


Figure 35: $osar_top$ Top-level entity of the Onput Segmentation and Reassembly (OSAR).

5.6 Control Cell Processor (CCP)

The Control Cell Processor (CCP) is the centralized control block of the system; hence, the CCP receives and transmits all control cells. A block diagram off the CCP is shown in Figure 36. Control cells are passed to the CCP from the ISAR via a FIFO input interface as described in Section 6.8. Response cells are passed from the CCP to the OSAR via a modified UTOPIA interface as described in Section 6.9. In order to conform to the convention that SPC-bound traffic from the FPX have a VPI of to 0x001, the CCP will set the VPI of response cells on the SPC_Control_VCI to 0x001. Note that all incoming control cells should have a VPI of 0x000. The CCP also timestamps all response cells with the upper 24-bits of a 32-bit counter. The counter is simply a running counter that increments once per clock cycle and rolls over.

The CCP provides control communication to several of the blocks in the system, allowing SPCs and other FPXs to exchange control information. The following sub-sections describe the control functions and associated control cell formats.

5.6.1 Register File Status & Updates

The CCP maintains the on-chip register file for the system. The control cell format for register file updates and status reads is shown in Figure 37. Description and default values for each register in the register file are shown in Table 1. Note that these default values are set by the reset signal. Also note that the Ibase_VCI must be an even multiple of four, as the actual sub-port VCI number is determined by appending the SPI to the upper 14-bits of the Ibase_VCI. Also note that the Obase_VCI must be an even multiple of eight, as the actual sub-port VCI number is determined by appending the PN to the upper 13-bits of the Obase_VCI. The associated OpCodes for register file operations are shown in Table 3. Note that the CCP generates an response cell with an OpCode value of one plus the original OpCode for each register file control cell.

5.6.2 System Counters and Flags

The CCP manages a number of system counters and flags for management and debugging purposes. The set of counters includes per-VCI input and output packet counters, as well as error counters and flags. All counters are 32-bits wide.

Table 4 lists the input packet counters with their associated number, name, and description. Table 5 lists the output packet counters with their associated number, name, and description. Table 6 lists the input control cell counters with their associated number, name, and description. Table 7 lists the output control cell counters with their associated number, name, and description. Table 9 lists the good cell counters with their associated number, name, and description.

Table 8 lists the packet and cell drop counters with their associated number, name, and description. Note that the ISAR cell drop counter counts the number of cells dropped at the ISAR due to congestion at the PSM or CARL interfaces. The ISAR invalid packet drop counter counts the number of packets dropped due to invalid AAL5 checksums, etc. The rare exception cases are captured in system flags which are defined as follows:

- **IPH (IP Header checksum fail)**: set by ISAR only if the AAL5 checksum passes and the IP header checksum fails; reset on counter/flag reset command.
- **LM (Length Mismatch)**: set by ISAR only if the AAL5 checksum passes and the IP Total Length and AAL5 length mismatch; reset on counter/flag reset command.

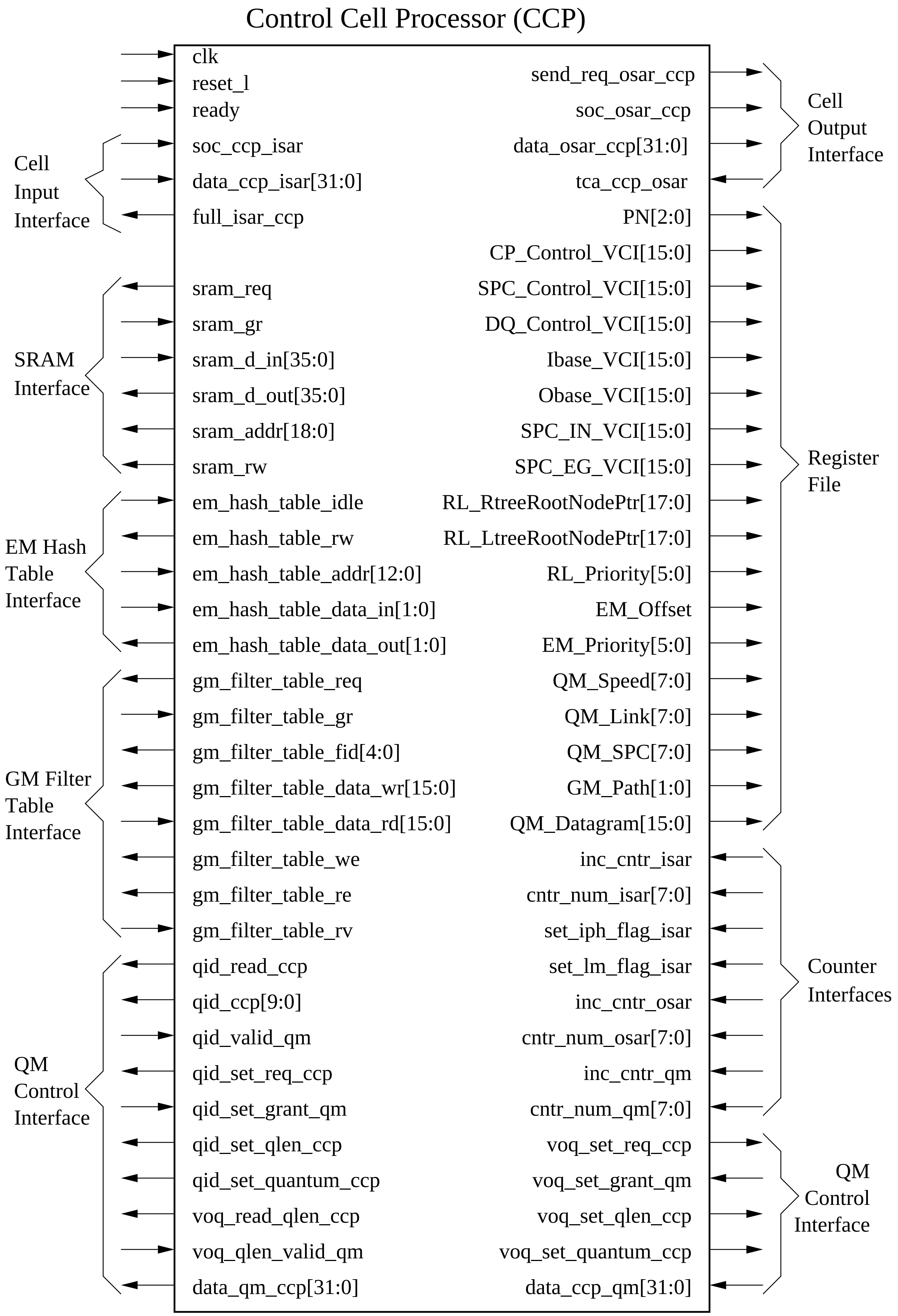


Figure 36: $ccp_{b}lock$ Block diagram of Control Cell Processor (CCP).

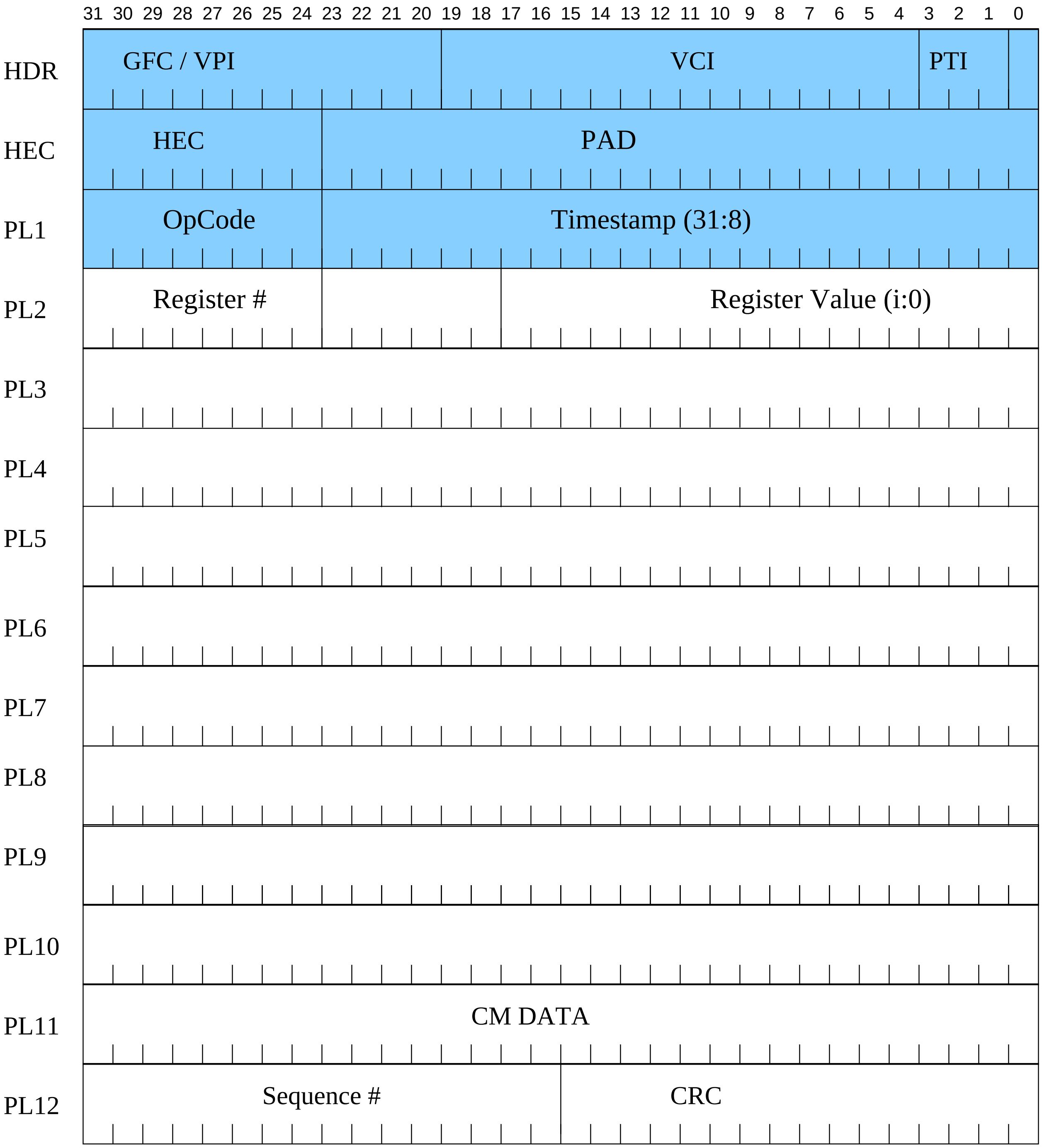


Figure 37: $reg_c c$ Control cell format for register file updates. Note that the width of the register value depends on the register number, where i is the width of the updated register minus one.

Table 1: Description and default values for register file. Registers are set to default values at system startup, hardware reset, and reset control command. *Note that response cells for commands received on the SPC_Control_VCI will be transmitted with a VPI = 0x001. **Note that rate specifications use floating point representation: $1.x \times 2^y$ where x = (upper 4-bits), y = (lower 4-bits)

Reg Number	*Reg Name*	*Width (bits)*	*Default*	*Description*
0x00	PN (Port Number)	3	0 (0x0)	Physical port number of the FPX
0x01	CP_Control_VCI	16	35 (0x23)	Control cell VCI for CP
0x02	SPC_Control_VCI	16	29 (0x1D)	Control cell VCI for SPC * Response cell VPI = 0x001
0x03	DQ_Control_VCI	16	61 (0x3D)	Control cell VCI for DQ
0x04	Ibase_VCI	16	128 (0x80)	Base VCI for link traffic
0x05	Obase_VCI	16	64 (0x40)	Base VCI for switch traffic
0x06	SPC_IN_VCI	16	62 (0x3E)	VCI for ingress SPC traffic
0x07	SPC_EG_VCI	16	63 (0x3F)	VCI for egress SPC traffic
0x08	RL_LtreeRootNodePtr	18	0 (0x0)	Root node pointer for the left subtree of the Split-Tree FIPL Tree Bitmap
0x1F	RL_RtreeRootNodePtr	18	2 (0x2)	Root node pointer for the right subtree of the Split-Tree FIPL Tree Bitmap
0x09	RL_Priority	6	60 (0x3C)	Priority of route lookup results
0x0A	EM_Offset	1	1	Address offset of exact match entries
0x0B	EM_Priority	6	56 (0x38)	Priority of exact match results
0x0C	QM_Speed	8	2 (0x02)	Speed advantage (ratio)
0x0D	QM_Link	8	x=.875, y=13 (0xED) (983 Mb/s)	Link rate in multiples of 64kb/s ** See note in caption
0x0E	QM_SPC	8	x=.5, y=11 (0x8B) (197 Mb/s)	SPC rate in multiples of 64kb/s ** See note in caption
0x0F	GM_Path	2	01	Active path for general match filters 00 = off, 01 = ingress 10 = egress, 11 = ingress & egress
0x10	QM_Datagram	15	2048 (0x0800)	Rate for datagram traffic

Table 2: Description of debugging registers in the TEMPORARY _debug versions of the design.

Reg Number	*Reg Name*	*Width (bits)*	*Default*	*Description*
0x10	ISAR/CARL	24	0 (0x0)	Counts header transfers from ISAR to CARL (soh_isar pulses)
0x11	CARL/QM	24	0 (0x0)	Counts header transfers from CARL to QM (wen_qm positive edges)
0x12	QM/OSAR(LC)	24	0 (0x0)	Counts header transfers from QM to OSAR(LC) (wen_lc_osar positive edges)
0x13	QM/OSAR(SW)	24	0 (0x0)	Counts header transfers from QM to OSAR(SW) (wen_sw_osar positive edges)
0x14	OSAR(SW)/PSM	24	0 (0x0)	Counts packet pointer transfers from OSAR(SW) to PSM (pkt_req_osar_sw pulses)
0x15	PSM/OSAR(SW)	24	0 (0x0)	Counts first packet chunk transfers from PSM to OSAR(SW) (sop_psm_sw pulses)
0x16	PSM/OSAR(SW)	24	0 (0x0)	Counts last packet chunk transfers from PSM to OSAR(SW) (eop_psm_sw pulses)
0x17	OSAR(LC)/PSM	24	0 (0x0)	Counts packet pointer transfers from OSAR(LC) to PSM (pkt_req_osar_lc pulses)
0x18	PSM/OSAR(LC)	24	0 (0x0)	Counts first packet chunk transfers from PSM to OSAR(LC) (sop_psm_lc pulses)
0x19	PSM/OSAR(LC)	24	0 (0x0)	Counts last packet chunk transfers from PSM to OSAR(LC) (eop_psm_lc pulses)

Table 3: OpCodes for register operations.

OpCode	*Operation*
0x00	Reset registers to default values
0x01	Reset registers response
0x02	Read specified register
0x03	Read specified register response
0x04	Write specified register
0x05	Write specified register response

Table 4: Input Packet Counter (IPC) table: per-VCI input packet counters incremented when a packet is received by ISAR on the associated VCI.

Counter Number	*Counter Name*	*Description*
0x00	SP0-IPC	Sub-Port 0 input packet counter
0x01	SP1-IPC	Sub-Port 1 input packet counter
0x02	SP2-IPC	Sub-Port 2 input packet counter
0x03	SP3-IPC	Sub-Port 3 input packet counter
0x04	PN0-IPC	Port Number 0 input packet counter
0x05	PN1-IPC	Port Number 1 input packet counter
0x06	PN2-IPC	Port Number 2 input packet counter
0x07	PN3-IPC	Port Number 3 input packet counter
0x08	PN4-IPC	Port Number 4 input packet counter
0x09	PN5-IPC	Port Number 5 input packet counter
0x0A	PN6-IPC	Port Number 6 input packet counter
0x0B	PN7-IPC	Port Number 7 input packet counter
0x0C	SPCI-IPC	Ingress SPC input packet counter
0x0D	SPCE-IPC	Egress SPC input packet counter

The OpCodes for system counter and flag commands are shown in Table 10, while the control cell format for system counter and flag operations is shown in Figure 38.

Counter increment commands are passed to the CCP from the other blocks of the design. These commands are simply an increment signal accompanied by a register number.

5.6.3 CARL Updates

The Route Lookup and Exact Filter Match blocks share an interface to the SRAM used by CARL. In order to update or read the contents of this memory for route updates or exact match filter updates, control cells must use the format shown in Figure 39. Note that this cell format must be used in order to read Exact Match packet and byte counters. Note that these commands may also be used to "dump" the contents of memory for debugging purposes.

Note that updating an Exact Match filter requires updating the contents of the on-chip hash table. The control cell format for these operations is shown in Figure 40. Upon receipt of the control cell, the CCP must request access to the EM hash table to prevent a simultaneous read/write operation which would result in undefined read output. After the grant signal is asserted, the CCP may access the EM hash table. Further details and timing diagrams are contained in Section 6.11.

Updating a General Match Filter requires updating the contents of the on-chip filter table. The control cell format for these operation is shown in Figure 41. A request/grant interface similar to the EM hash table exists for the GM filter table. Further details and timing diagrams are contained in Section 6.11.

The associated opcodes for control cells and response cells are given in Table 11.

5.6.4 Queue Status and Updates

In order to monitor and control the behavior of the Queue Manger, software must have the ability to read queue lengths, transmit queue length information, and update scheduling quantum. The CCP/QMGR control

<table>
<tr><th></th><th>31</th><th>30</th><th>29</th><th>28</th><th>27</th><th>26</th><th>25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>HDR</td><td colspan="12">GFC / VPI</td><td colspan="16">VCI</td><td colspan="3">PTI</td><td></td></tr>
<tr><td>HEC</td><td colspan="8">HEC</td><td colspan="24">PAD</td></tr>
<tr><td>PL1</td><td colspan="8">OpCode</td><td colspan="24">Timestamp (31:8)</td></tr>
<tr><td>PL2</td><td colspan="8">Counter #</td><td colspan="22"></td><td>L
M</td><td>I
P
H</td></tr>
<tr><td>PL3</td><td colspan="32">Counter Value (31:0)</td></tr>
<tr><td>PL4</td><td colspan="32"></td></tr>
<tr><td>PL5</td><td colspan="32"></td></tr>
<tr><td>PL6</td><td colspan="32"></td></tr>
<tr><td>PL7</td><td colspan="32"></td></tr>
<tr><td>PL8</td><td colspan="32"></td></tr>
<tr><td>PL9</td><td colspan="32"></td></tr>
<tr><td>PL10</td><td colspan="32"></td></tr>
<tr><td>PL11</td><td colspan="32">CM DATA</td></tr>
<tr><td>PL12</td><td colspan="16">Sequence #</td><td colspan="16">CRC</td></tr>
</table>

Figure 38: $ccp_{c}ounter_{r}ead$ Format of control cells used to read system counters. Note that system flags are included in every counter read response.

Table 5: Output Packet Counter (OPC) table: per-VCI output packet counters incremented when a packet is transmitted by OSAR on the associated VCI.

Counter Number	*Counter Name*	*Description*
0x10	SP0-OPC	Sub-Port 0 output packet counter
0x11	SP1-OPC	Sub-Port 1 output packet counter
0x12	SP2-OPC	Sub-Port 2 output packet counter
0x13	SP3-OPC	Sub-Port 3 output packet counter
0x14	PN0-OPC	Port Number 0 output packet counter
0x15	PN1-OPC	Port Number 1 output packet counter
0x16	PN2-OPC	Port Number 2 output packet counter
0x17	PN3-OPC	Port Number 3 output packet counter
0x18	PN4-OPC	Port Number 4 output packet counter
0x19	PN5-OPC	Port Number 5 output packet counter
0x1A	PN6-OPC	Port Number 6 output packet counter
0x1B	PN7-OPC	Port Number 7 output packet counter
0x1C	SPCI-OPC	Ingress SPC output packet counter
0x1D	SPCE-OPC	Egress SPC output packet counter

Table 6: Input Control cell Counter (ICC) table: per-VCI input control cell counters incremented when a control cell is received by the ISAR on the associated VCI.

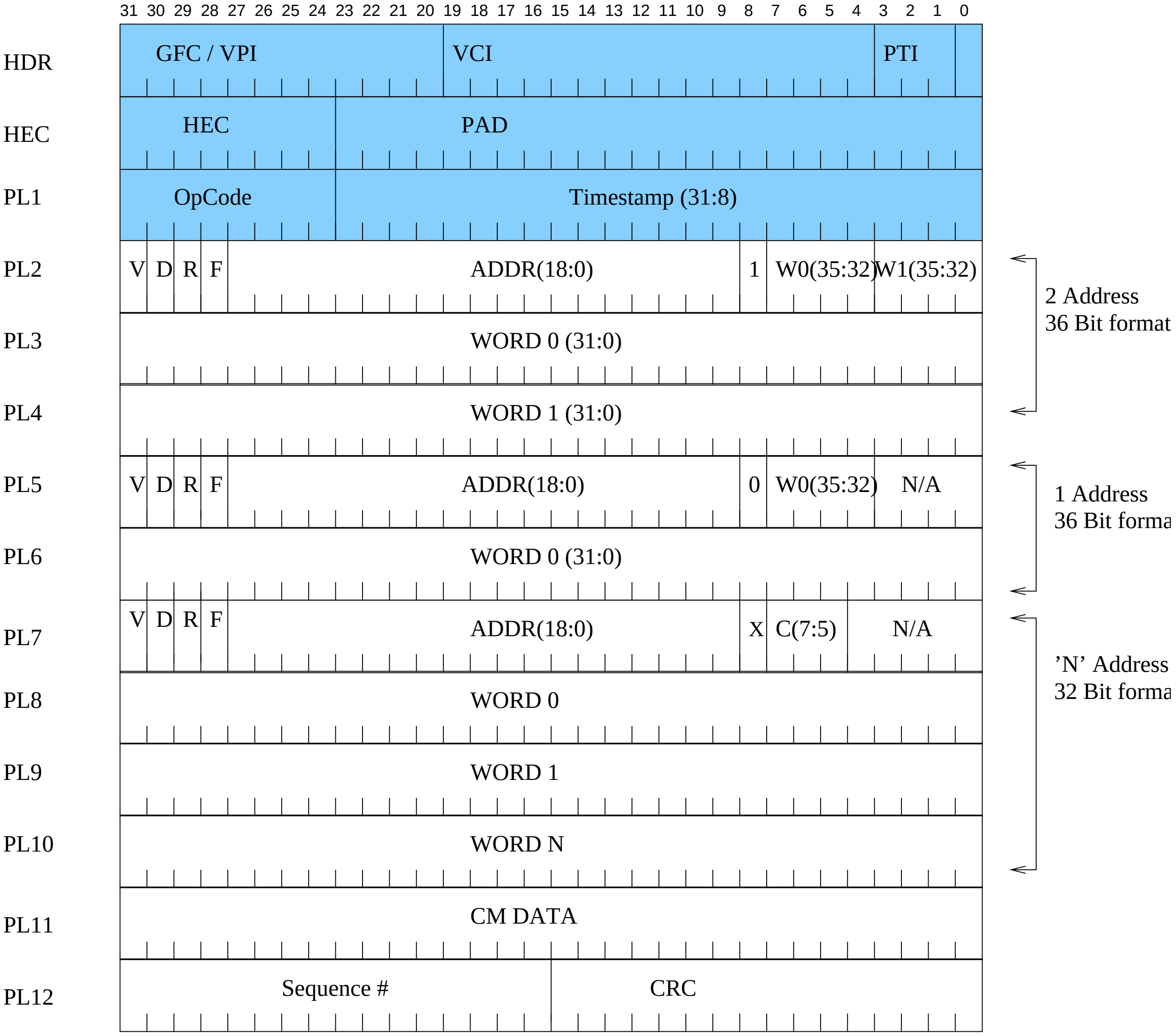
Counter Number	*Counter Name*	*Description*
0x20	SPC-ICC	SPC input control cell counter
0x21	DQ-ICC	DQ input control cell counter
0x22	CP-ICC	CP input control cell counter

Table 7: Output Control cell Counter (OCC) table: per-VCI output control cell counters incremented when a control cell is transmitted by the OSAR on the associated VCI.

Counter Number	*Counter Name*	*Description*
0x30	SPC-OCC	SPC output control cell counter
0x31	DQ-OCC	DQ output control cell counter
0x32	CP-OCC	CP output control cell counter

Table 8: Drop Counter (DPC) table: counters incremented when packets or cells are dropped.

Counter Number	*Counter Name*	*Description*
0x40	IIC-DPC-LC	ISAR input cell drop counter (Ingress Side)
0x45	IIC-DPC-SW	ISAR input cell drop counter (Egress Side)
0x41	IVP-DPC-LC	ISAR invalid packet drop counter (Ingress Side)
0x46	IVP-DPC-SW	ISAR invalid packet drop counter (Egress Side)
0x42	QMLC-DPC	QM packet drop counter for LC queues
0x43	QMSW-DPC	QM packet drop counter for SW queues
0x44	QMSPC-DPC	QM packet drop counter for SPC queues



V − Valid Command:	1 = Valid command, 0 = Invalid, EOC
D − Device:	1 = Device 1, 0 = Device 0
R − Read or Write:	1 = Read, 0 = Write
F − 32 or 36 bit:	1 = 36 bit, 0 = 32 bit

Figure 39: $sram_u pdate_c c$ Control cell format for CARL SRAM updates.

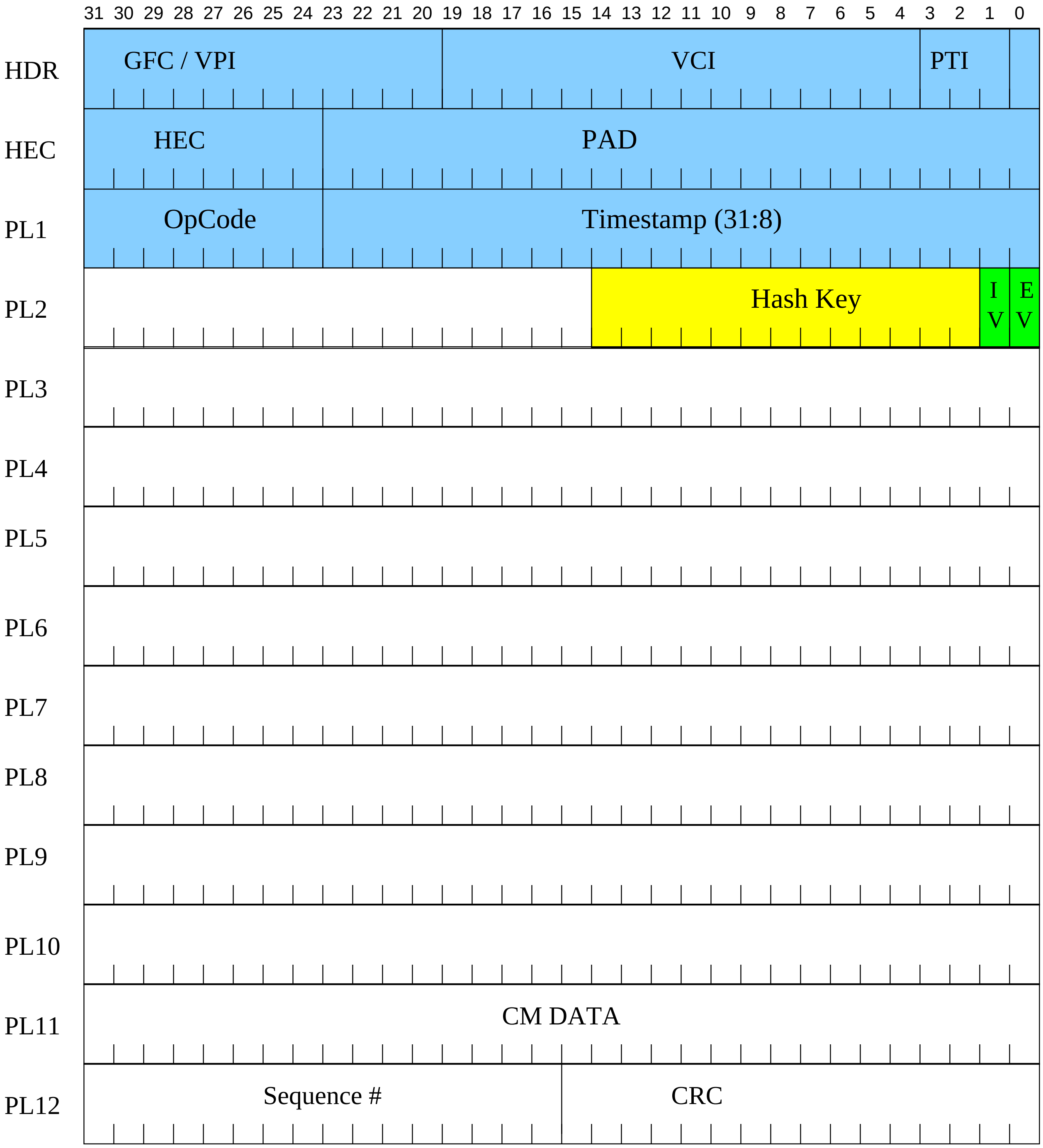
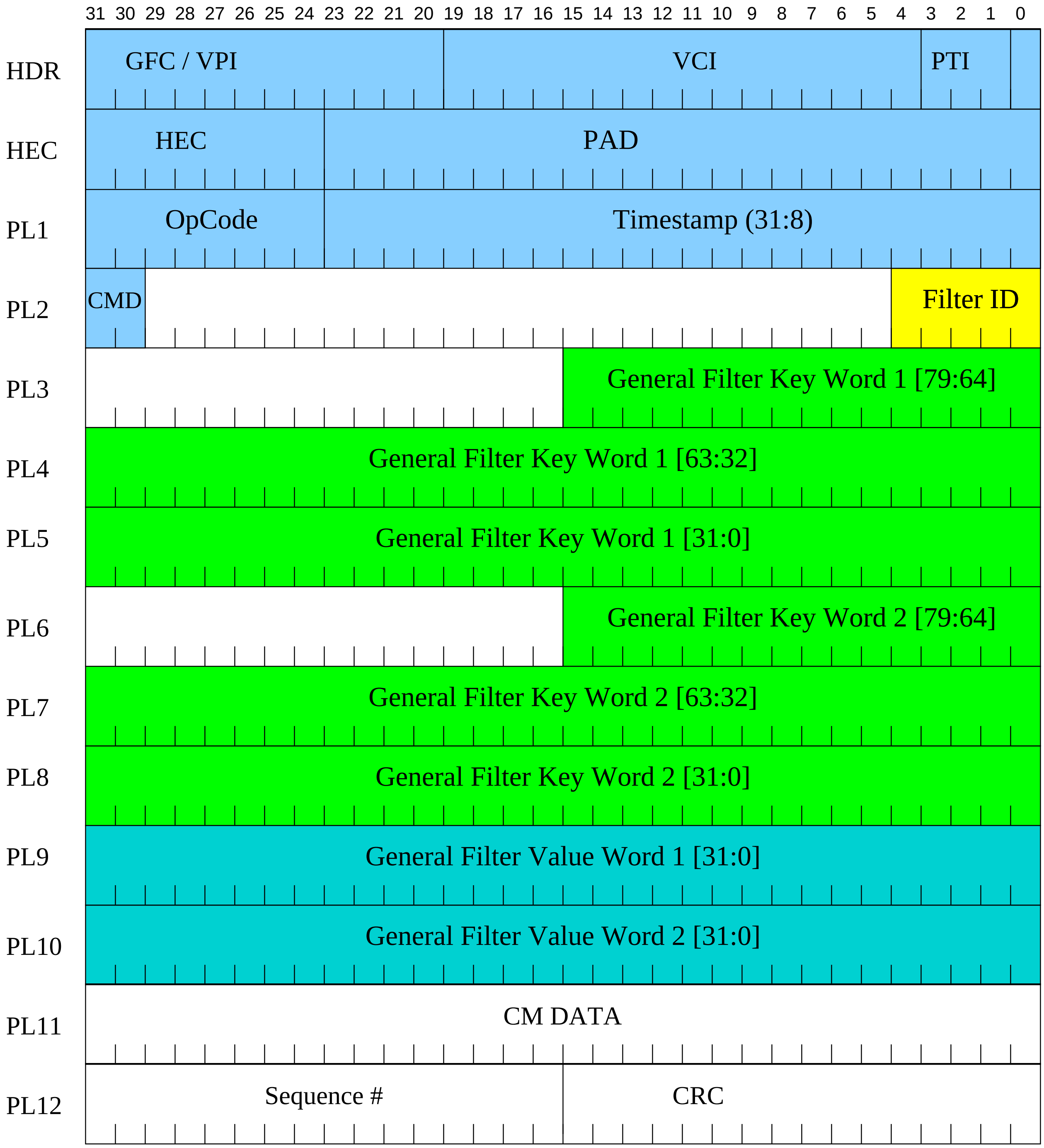


Figure 40: $hash_{t}able_{u}pdate_{c}c$ Control cell format for EM hash table updates.



HDR	GFC / VPI	VCI	PTI	
HEC	HEC	PAD		
PL1	OpCode	Timestamp (31:8)		
PL2	CMD		Filter ID	
PL3		General Filter Key Word 1 [79:64]		
PL4	General Filter Key Word 1 [63:32]			
PL5	General Filter Key Word 1 [31:0]			
PL6		General Filter Key Word 2 [79:64]		
PL7	General Filter Key Word 2 [63:32]			
PL8	General Filter Key Word 2 [31:0]			
PL9	General Filter Value Word 1 [31:0]			
PL10	General Filter Value Word 2 [31:0]			
PL11	CM DATA			
PL12	Sequence #	CRC		

CMD = 0x0 (No operation)
CMD = 0x1 (Write table entry)
CMD = 0x2 (Read table entry)

Figure 41: $gm_{f}ilter_{u}pdate_{c}c$ Control cell format for GM filter table updates.

Table 9: ISAR Good Cell Counters.

Counter Number	*Counter Name*	*Description*
0x50	IIC-GDC-LC	ISAR good received cell counter (Ingress Side)
0x51	IIC-GDC-SW	ISAR good received cell counter (Egress Side)

Table 10: OpCodes for system counter and flag functions.

OpCode	*Action (parameters)*
0x10	Reset system counters/flags
0x11	Reset system counters/flags response
0x12	Read specified counter
0x13	Read specified counter response

interface provides these capabilities. The OpCodes for the various operations are shown in Table 12.

The QID Queue Length Query command allows components to read the length of a specific flow queue identified by its Queue Identifier (QID). The control cell format for the QID Queue Length Query is shown in Figure 42.

The QID Quantum Update command allows components to update the 32-bit quantum information (weight, or byte count) used by the Queue Manger for scheduling the queue identified by the QID. The control cell format for the QID Quantum Updates is shown in Figure 43.

The QID Queue Length Threshold Update command allows components to update the 32-bit queue length threshold (bytes) for a specific QID in the Queue Manager. The control cell format for the QID Queue Length Threshold Update is shown in Figure 44.

The VOQ Queue Length Query command allows components to read the length of the eight Virtual Output Queues (VOQs) and the sum of all egress queues. All lengths are 32-bit byte counts. The control cell format for the VOQ Queue Length Query is shown in Figure 45.

The VOQ Quantum Update command allows components to update the quantum information (weight, or byte count) used by the Queue Manger for scheduling. VOQ quantum are 32-bit byte counts with a minimum equal to the maximum packet size (MTU of the egress port). Note that this command also updates the switch bandwidth, given as a 32-bit rate in Kbps. The control cell format for the VOQ Quantum Update command is shown in Figure 46.

The VOQ Queue Length Threshold Update command allows components to update the queue length

Table 11: OpCodes for Route Lookup SRAM update functions.

OpCode	*Action (parameters)*
0x40	SRAM update
0x41	SRAM update response
0x42	EM hash table read
0x43	EM hash table read response
0x44	EM hash table write
0x45	EM hash table write response
0x46	GM filter table update
0x47	GM filter table update response

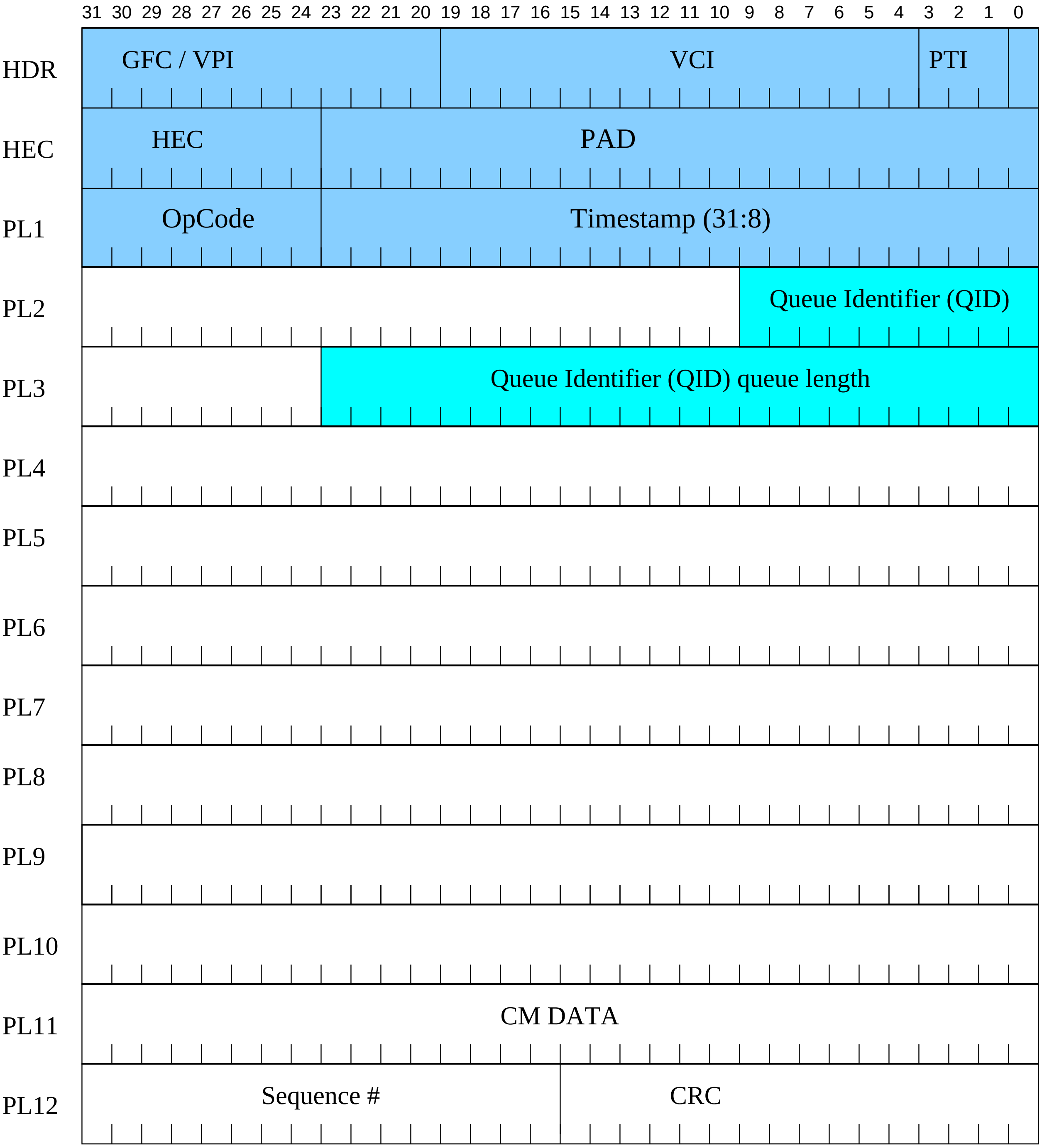


Figure 42: $qlen_{q}uery_{c}c$ Control cell format for QID Queue Length Query (OpCode 0x60).

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HDR	GFC / VPI												VCI																PTI			
HEC	HEC								PAD																							
PL1	OpCode								Timestamp (31:8)																							
PL2																							Queue Identifier (QID)									
PL3	Queue Identifier (QID) quantum (bytes)																															
PL4																																
PL5																																
PL6																																
PL7																																
PL8																																
PL9																																
PL10																																
PL11	CM DATA																															
PL12	Sequence #																CRC															

Figure 43: $qid_{q}uant_{u}pdate_{c}c$ Control cell format for QID Quantum Updates (OpCode 0x62).

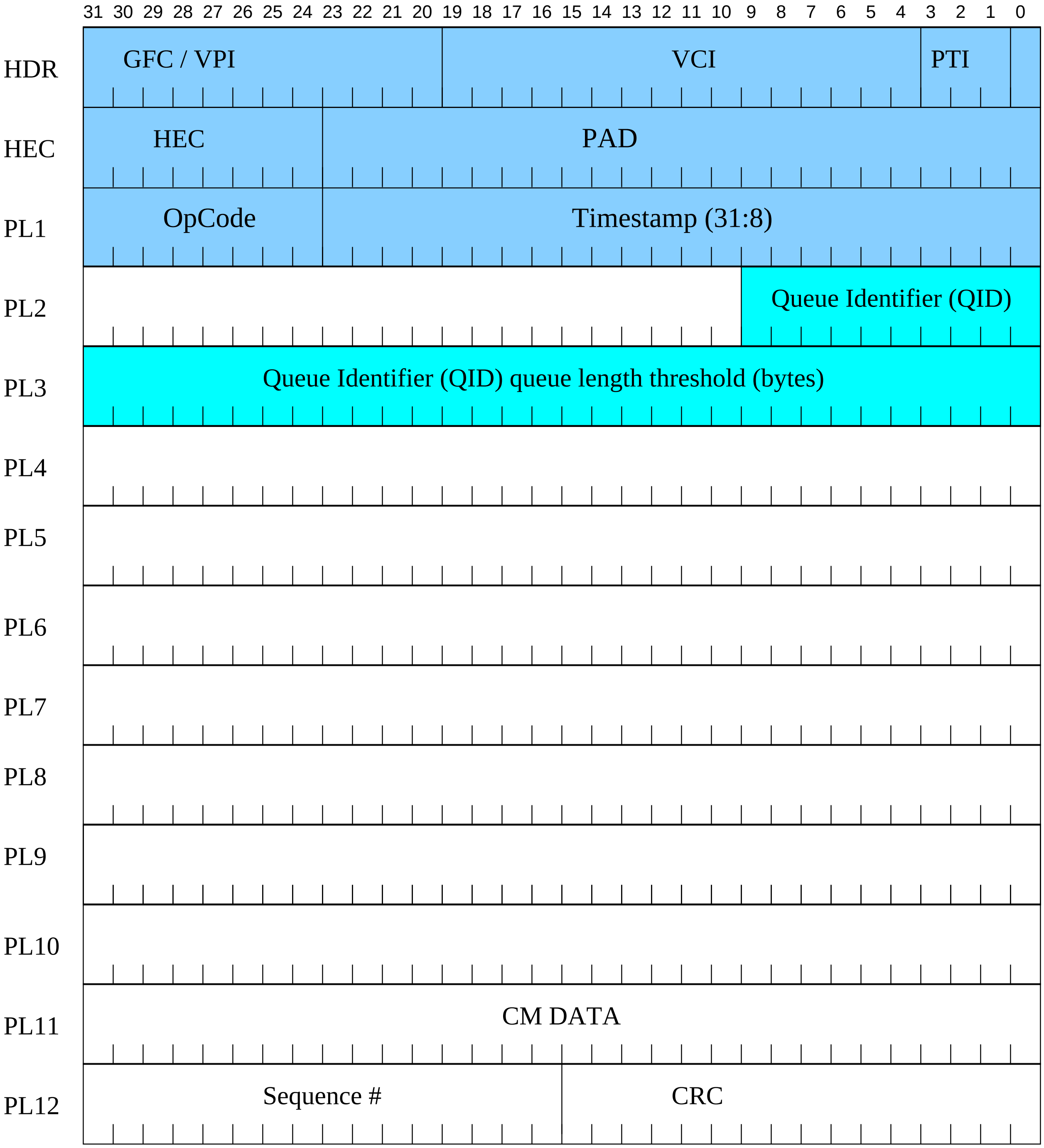
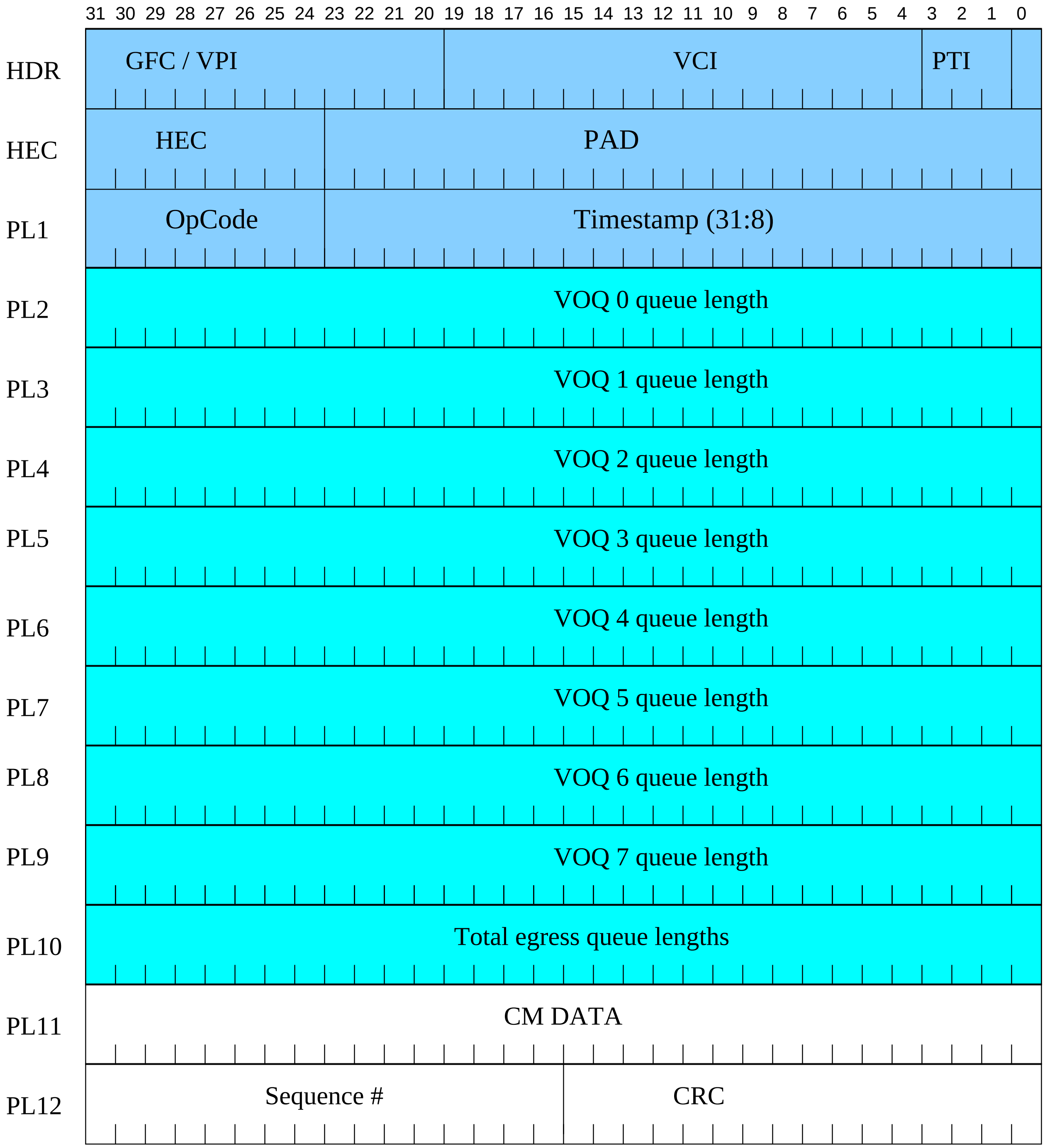
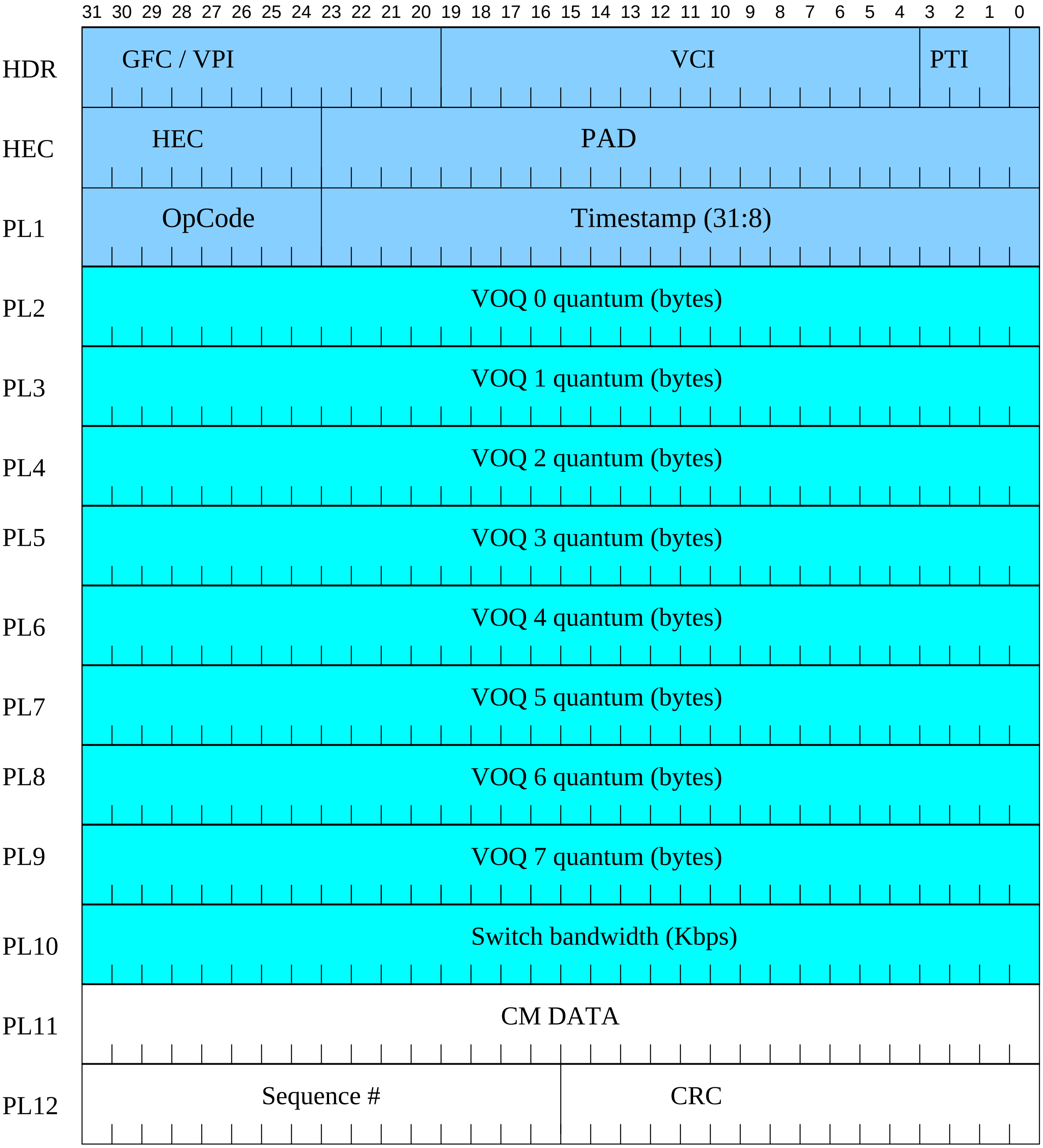


Figure 44: $qid_{q}len_{u}pdate_{c}c$ Control cell format for QID Queue Length Updates (OpCode 0x64).



<table>
<tr><th></th><th>31</th><th>30</th><th>29</th><th>28</th><th>27</th><th>26</th><th>25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>HDR</td><td colspan="12">GFC / VPI</td><td colspan="16">VCI</td><td colspan="3">PTI</td><td></td></tr>
<tr><td>HEC</td><td colspan="8">HEC</td><td colspan="24">PAD</td></tr>
<tr><td>PL1</td><td colspan="8">OpCode</td><td colspan="24">Timestamp (31:8)</td></tr>
<tr><td>PL2</td><td colspan="32">VOQ 0 queue length</td></tr>
<tr><td>PL3</td><td colspan="32">VOQ 1 queue length</td></tr>
<tr><td>PL4</td><td colspan="32">VOQ 2 queue length</td></tr>
<tr><td>PL5</td><td colspan="32">VOQ 3 queue length</td></tr>
<tr><td>PL6</td><td colspan="32">VOQ 4 queue length</td></tr>
<tr><td>PL7</td><td colspan="32">VOQ 5 queue length</td></tr>
<tr><td>PL8</td><td colspan="32">VOQ 6 queue length</td></tr>
<tr><td>PL9</td><td colspan="32">VOQ 7 queue length</td></tr>
<tr><td>PL10</td><td colspan="32">Total egress queue lengths</td></tr>
<tr><td>PL11</td><td colspan="32">CM DATA</td></tr>
<tr><td>PL12</td><td colspan="16">Sequence #</td><td colspan="16">CRC</td></tr>
</table>

Figure 45: $voq_qlen_query_cc$ Control cell format for VOQ Queue Length Query (OpCode 0x66).



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HDR	GFC / VPI												VCI																PTI			
HEC	HEC								PAD																							
PL1	OpCode								Timestamp (31:8)																							
PL2	VOQ 0 quantum (bytes)																															
PL3	VOQ 1 quantum (bytes)																															
PL4	VOQ 2 quantum (bytes)																															
PL5	VOQ 3 quantum (bytes)																															
PL6	VOQ 4 quantum (bytes)																															
PL7	VOQ 5 quantum (bytes)																															
PL8	VOQ 6 quantum (bytes)																															
PL9	VOQ 7 quantum (bytes)																															
PL10	Switch bandwidth (Kbps)																															
PL11	CM DATA																															
PL12	Sequence #																CRC															

Figure 46: $voq_q uant_u pdate_c c$ Control cell format for VOQ Quantum Updates (OpCode 0x68).

Table 12: OpCodes for queue length status queries and updates.

OpCode	*Action (parameters)*
0x60	QID Queue Length Query
0x61	QID Queue Length Query response
0x62	QID Quantum Update
0x63	QID Quantum Update response
0x64	QID Queue Length Threshold Update
0x65	QID Queue Length Threshold Update response
0x66	VOQ Queue Lengths Query
0x67	VOQ Queue Lengths Query response
0x68	VOQ Quantum Updates
0x69	VOQ Quantum Updates response
0x70	VOQ Queue Length Thresholds Update
0x71	VOQ Queue Length Thresholds Update response

thresholds for the eight Virtual Output Queues (VOQs). The control cell format for the VOQ Queue Length Updates is shown in Figure 47.

<table>
<tr><th></th><th>31</th><th>30</th><th>29</th><th>28</th><th>27</th><th>26</th><th>25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>HDR</td><td colspan="12">GFC / VPI</td><td colspan="16">VCI</td><td colspan="3">PTI</td><td></td></tr>
<tr><td>HEC</td><td colspan="8">HEC</td><td colspan="24">PAD</td></tr>
<tr><td>PL1</td><td colspan="8">OpCode</td><td colspan="24">Timestamp (31:8)</td></tr>
<tr><td>PL2</td><td colspan="32">VOQ 0 queue length threshold (bytes)</td></tr>
<tr><td>PL3</td><td colspan="32">VOQ 1 queue length threshold (bytes)</td></tr>
<tr><td>PL4</td><td colspan="32">VOQ 2 queue length threshold (bytes)</td></tr>
<tr><td>PL5</td><td colspan="32">VOQ 3 queue length threshold (bytes)</td></tr>
<tr><td>PL6</td><td colspan="32">VOQ 4 queue length threshold (bytes)</td></tr>
<tr><td>PL7</td><td colspan="32">VOQ 5 queue length threshold (bytes)</td></tr>
<tr><td>PL8</td><td colspan="32">VOQ 6 queue length threshold (bytes)</td></tr>
<tr><td>PL9</td><td colspan="32">VOQ 7 queue length threshold (bytes)</td></tr>
<tr><td>PL10</td><td colspan="32"></td></tr>
<tr><td>PL11</td><td colspan="32">CM DATA</td></tr>
<tr><td>PL12</td><td colspan="16">Sequence #</td><td colspan="16">CRC</td></tr>
</table>

Figure 47: $voq_qlen_updatec_cc$ Control cell format for VOQ Queue Length Threshold Updates (OpCode 0x70).

6 Interfaces

This section documents interfaces with components outside of RAD and interfaces among modules within the RAD. Signal names, formats and timing diagram are provided.

6.1 Line Card (LC) and Switch (SW) Interfaces

Packets are received from or transmitted to one of the Sub-Ports (SPs) of the Line Card as AAL5 encapsulated frames as shown in Figure 48. Note that the NSP shim is removed prior to transmission.

Packets are received from or transmitted to the switch fabric as AAL5 encapsulated frames *with* an NSP InterPort shim inserted prior to the IP header as shown in Figure 49.

Packets are received from or transmitted to the SPC for full packet processing as AAL5 encapsulated frames *with* an NSP IntraPort shim inserted prior to the IP header as shown in Figure 50.

AAL5 frames are segmented across multiple ATM cells. Therefore, the NID Line Card (LC), NID Switch (SW), and Control Cell Processor (CCP) interfaces use a 32-bit UTOPIA interface. The specification of this interface is identical to the cell I/O interface specification for RAD modules given in [5]. Please refer to that document for further details. The timing diagram for cell input is shown in Figure 51. The timing diagram for cell output is shown in Figure 52. Note the specification for Transmit Cell Available (TCA) assertion and sampling. This specification prevents cell loss across the NID/RAD interface.

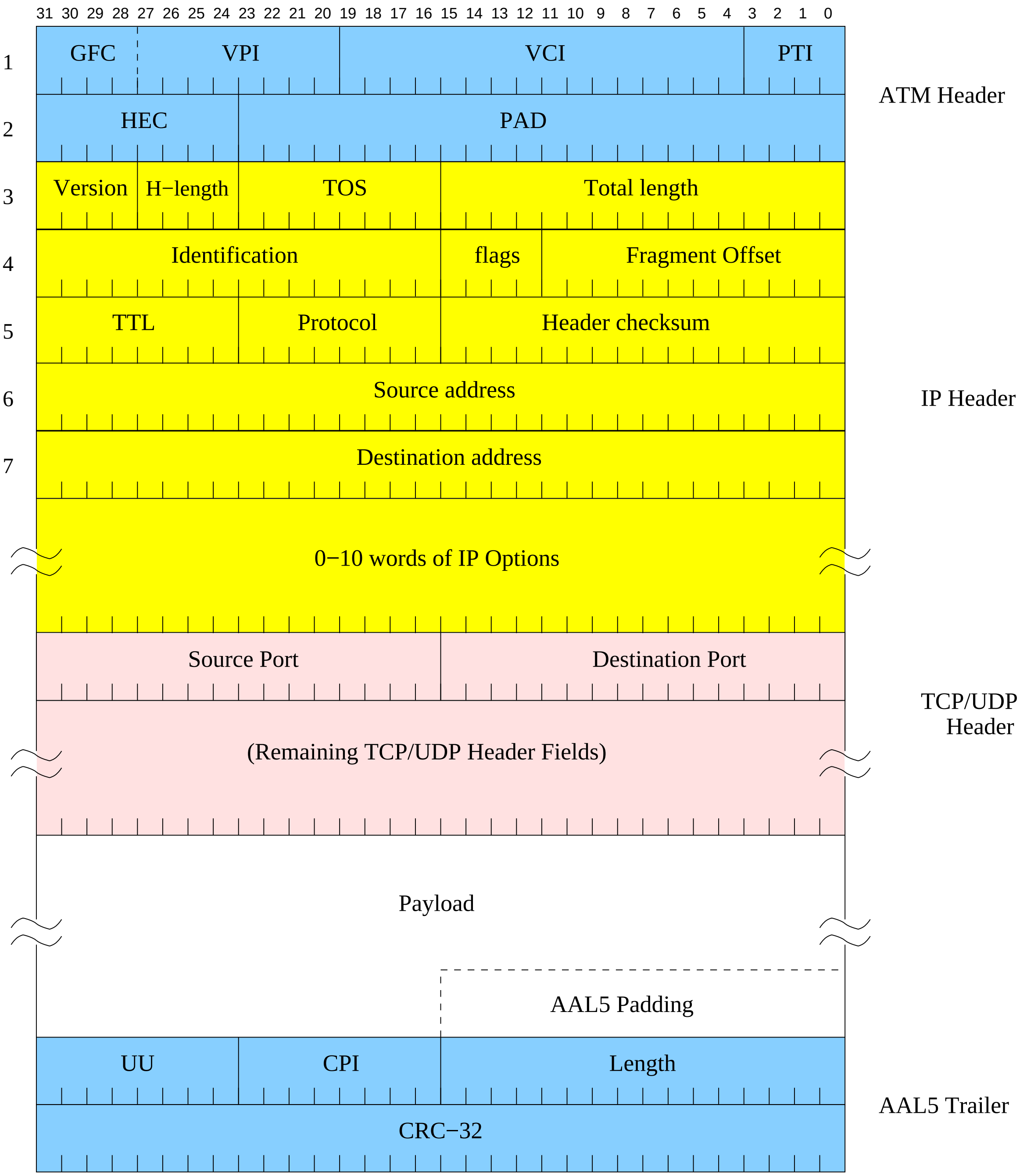


Figure 48: $aal5_ip_tcp$ Format of packets received from and transmitted to one of the Sub-Ports (SPs) via the NID Line Card (LC) interface. Packets are encapsulated in AAL5 frames whose individual cells transit a 32-bit UTOPIA interface.

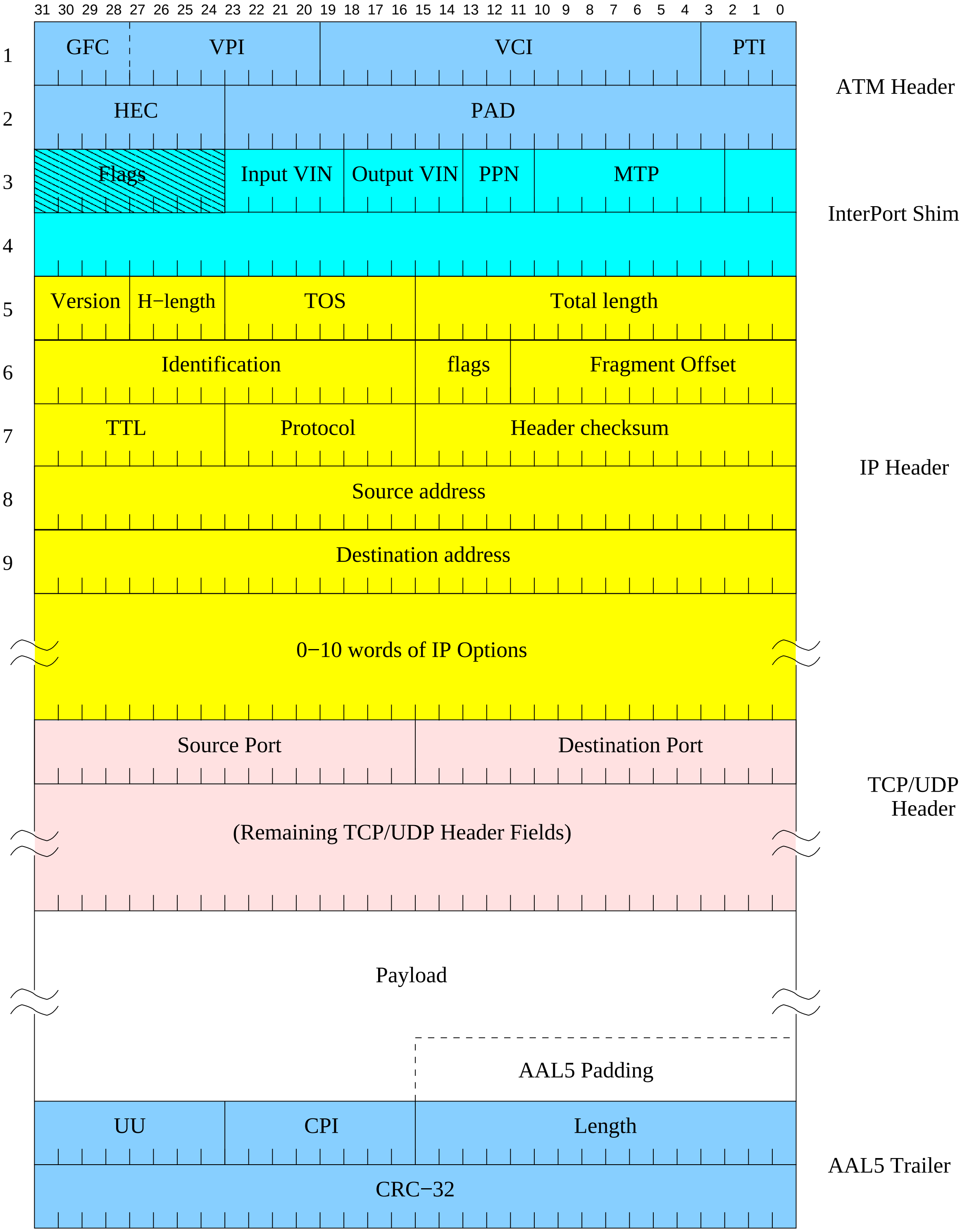


Figure 49: $msr_{i}p_{p}acket$ Format of packets received from and transmitted to the switch fabric as AAL5 encapsulated frames across the 32-bit UTOPIA interface. Note that the NSP InterPort Shim resides between the ATM cell header and the IPv4 header.

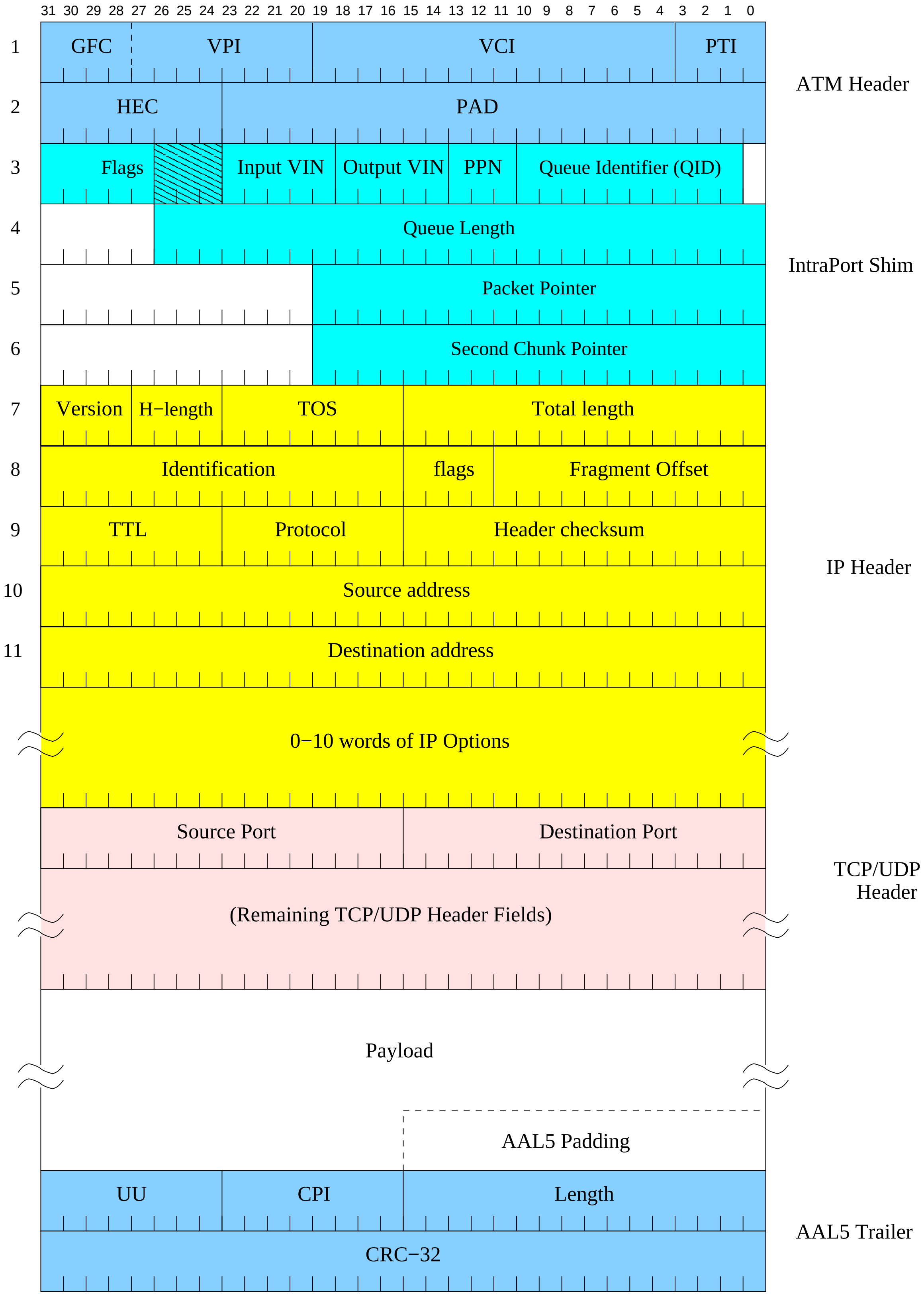


Figure 50: spc_frame Format of AAL5 frame sent to and received from the SPC for packet processing.

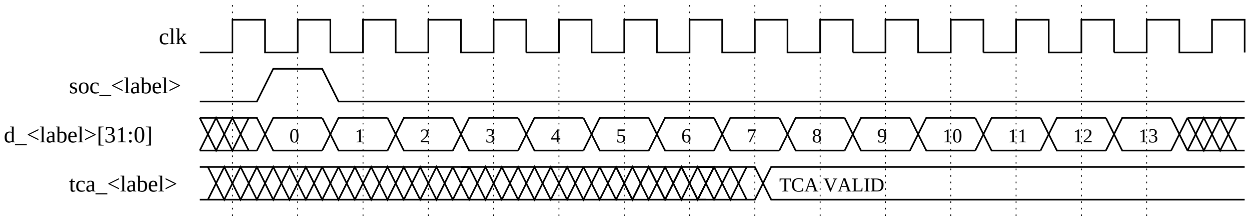


Figure 51: $cell_input_timing$ Timing diagram for cell input on the RAD FPGA of the FPX. All NSP components receiving cells conform to the cell input specification for RAD modules.

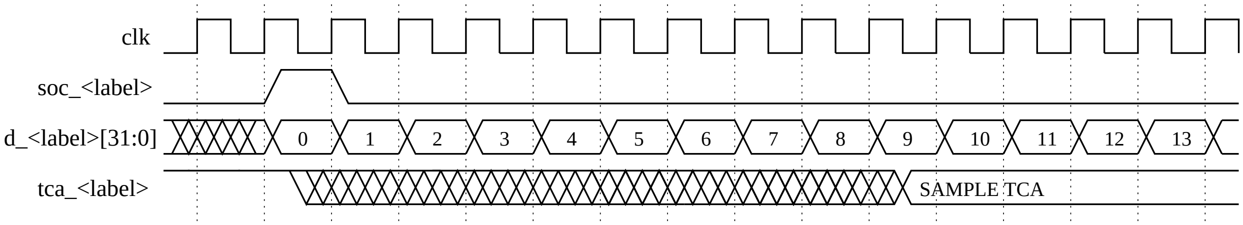


Figure 52: $cell_output_timing$ Timing diagram for cell output on the RAD FPGA of the FPX. All NSP components transmitting cells conform to the cell output specification for RAD modules.

6.2 ISAR/PSM Interface

The Input Segmentation and Reassembly (ISAR) block interfaces to the input side of the Packet Storage Manager (PSM). Packets arriving from both the LC and SW interfaces are buffered in the ISAR and transferred to the PSM in fixed size chunks. The first chunk of a packet carries up to 120 bytes of the packet. All other chunks carry up to 124 bytes of the remaining packet. Figure 53 and Figure 54 shows the slight difference in format between chunks used as the first chunk and the n^{th} chunk. The first byte is reserved. The next 3 bytes of a chunk is used by the PSM to store the next chunk pointer. The lowest 5 bits of the 5^{th} byte of the first chunk of a packet is used to store the Input Virtual Identification Number (IVIN).

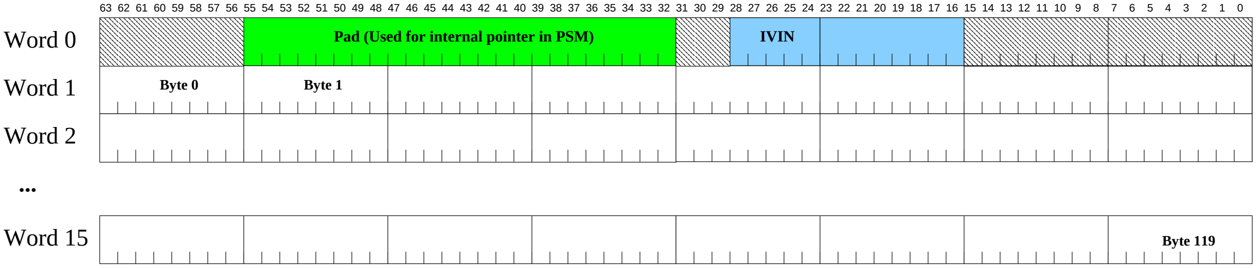


Figure 53: psm_chunk_1 Format of first data chunk of a packet sent to the Packet Storage Manager (PSM) from the Input Segmentation and Reassembly (ISAR) block.

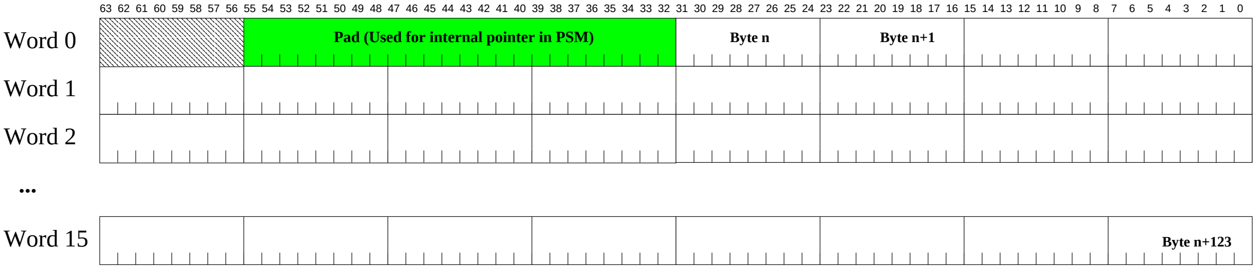


Figure 54: psm_chunk_n Format of remaining data chunks of a packet sent to the Packet Storage Manager (PSM) from the Input Segmentation and Reassembly (ISAR) block.

Chunks are transferred across a 64-bit data bus with associated control signals. The timing diagram for this interface is shown in Figure 55. Each signal is explained below:

- *packet_ptr_enq_isar* enqueues a packet pointer (the address of the first chunk for the packet).
- *packet_ptr_psm[31:0]* returns the packet pointer to ISAR several cycles after the *eop_isar* is asserted. PSM will return *packet_ptr*s in the same order the last chunks (flagged by *eop_isar*) are released to PSM.
- *no_free_chunk_psm* indicates that PSM runs out of free chunk list.
- *packet_length_isar[15:0]* indicates the length in bytes of packet whose first chunk is being transfered. Please note that this first could also be last chunk.
- *cid_isar[3:0]* indicates upto 16 context IDs to be supported by each PSM (can support upto 32 total, 16 ingress & 16 egress). Each context ID corresponds to one VCI value.

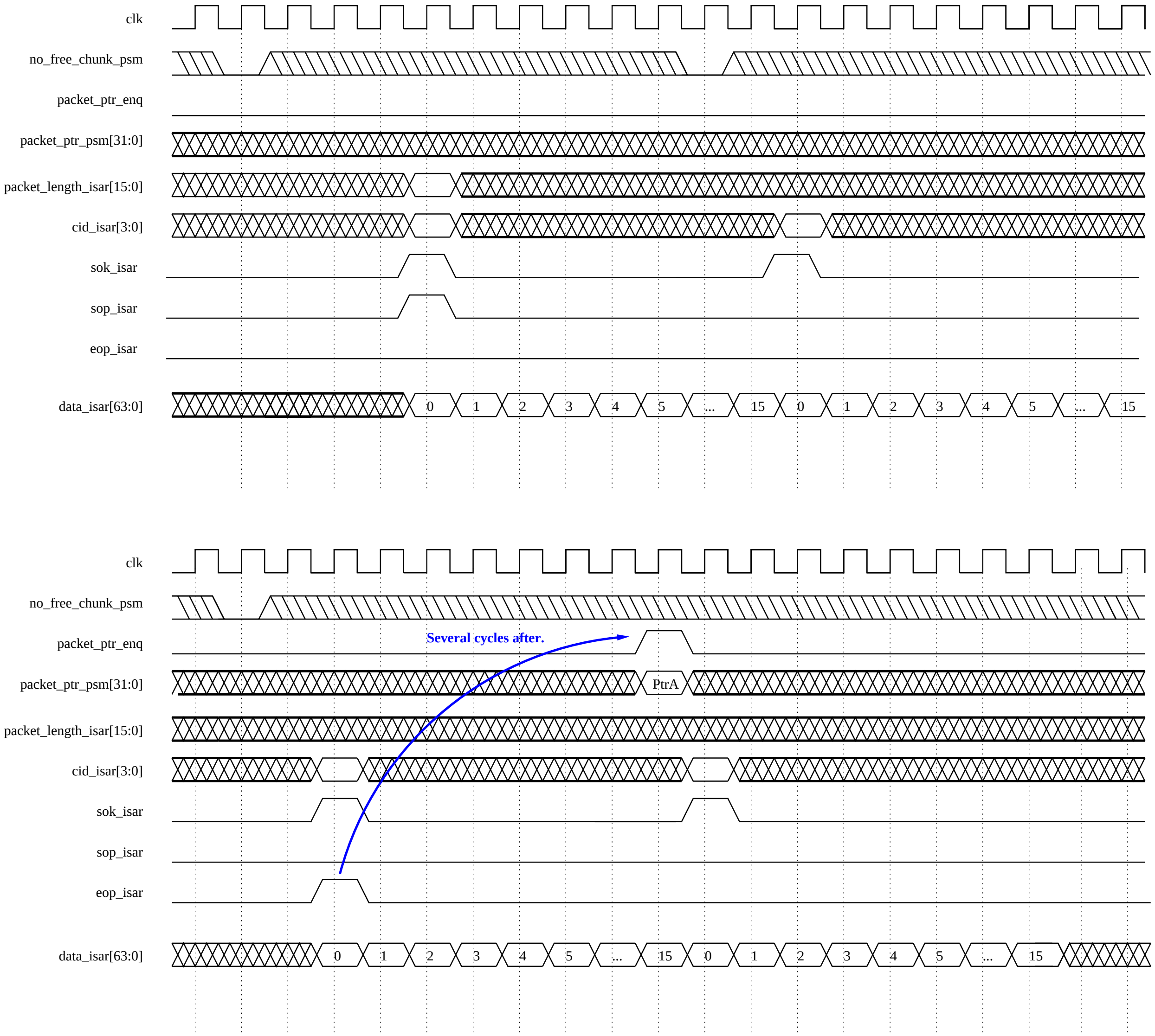


Figure 55: $isar_{p}sm_{t}iming$ Timing diagram for the ISAR/PSM interface.

- *sop_isar* flags the **s**tart **o**f an IP **p**acket.
- *eop_isar* flags the **e**nd **o**f an IP **p**acket.
- *ow_chunk_isar* informs PSM that the incoming chunk must **o**ver **w**rite previously written chunk at location specified by *chunk_addr_isar[31:0]*.
- *chunk_addr_isar[31:0]* specifies the location where the chunk should be written to whenever *ow_chunk_isar* is asserted.
- *data_valid* indicates that the *data_isar* signal is valid.
- *data_isar[63:0]* carries the chunk itself.

For a new packet, ISAR reserves a chunk from PSM, passing along the packet length and the corresponding Context ID (CID). Returned address to this chunk is used as the *packet_ptr*. Transfer of a chunk belonging to this packet will be flagged (during the first cycle of the chunk transfer) to indicate if a chunk is the first (*sop_isar* high), the last (*eop_isar* high), neither first nor last, or both first and last (*sop_isar* and *eop_isar* high) of the packet.

When *no_free_chunk_psm* is asserted high, ISAR should not release a chunk to PSM.

6.3 ISAR/CARL Interface

Packet references are transmitted from the Input Segmentation and Reassembly (ISAR) block to the Classification and Route Lookup (CARL) block once the entire packet has been received. The data frame format for the ISAR/CARL interface is shown in Figure 56 with timing relationship shown in Figure 57.

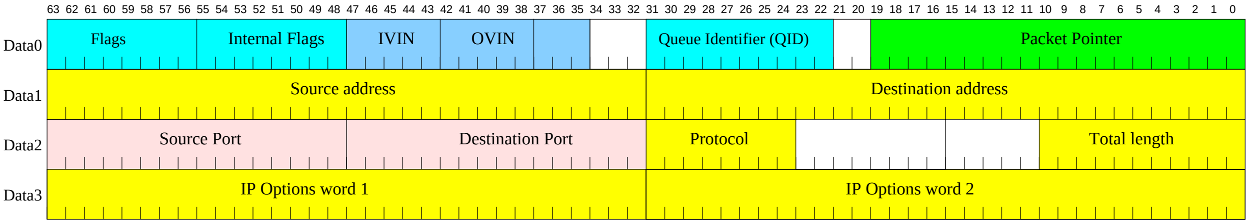


Figure 56: $isar_{c}arl_{f}rame$ Data frame format for ISAR/CARL interface.

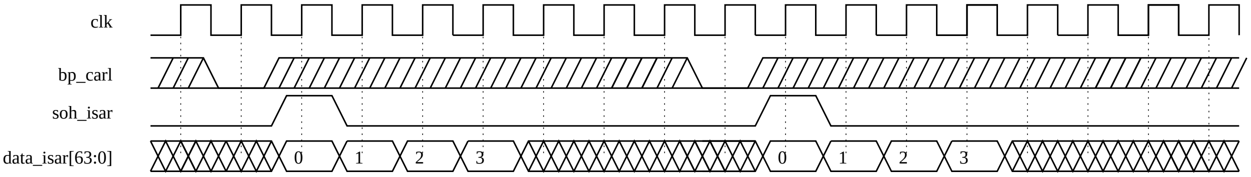


Figure 57: $isar_{c}arl_{t}iming$ Timing diagram for ISAR/CARL interface.

- *bp_carl* is used by CARL to back pressure ISAR. Back pressure is done on a per packet basis, i.e., once transfer of a packet header is initiated, it has to be completed regardless of the state of bp_carl signal.
- *data_isar[63:0]* carries the packet header as defined in Figure 56.
- *soh_isar* indicates the beginning of 4 consecutive cycles transfer of a packet header from ISAR to CARL.

Transfer may occur back-to-back.

6.4 CARL/QMGR Interface

Following classification and priority resolution. packet references are transmitted from the Classification and Route Lookup (CARL) block to the Queue Manager (QMGR) block via QM_IN_FIFO. The data frame format for the CARL/QMGR interface is shown in Figure 58 with timing relationship shown in Figure 59. *soh_qm_carl* indicates the beginning of 4 consecutive cycles transfer of a packet reference in *data_qm_carl[31:0]*. Transfer may occur back-to-back.

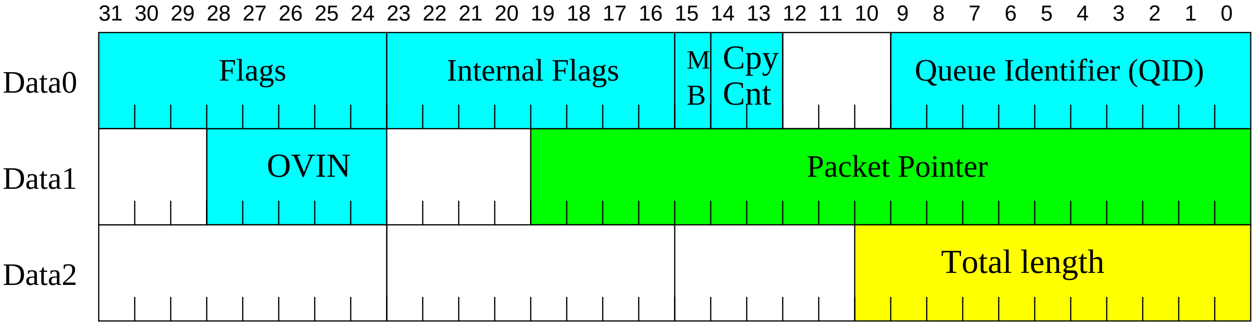


Figure 58: $carl_qm_frame$ Data frame format for CARL/QMGR interface.

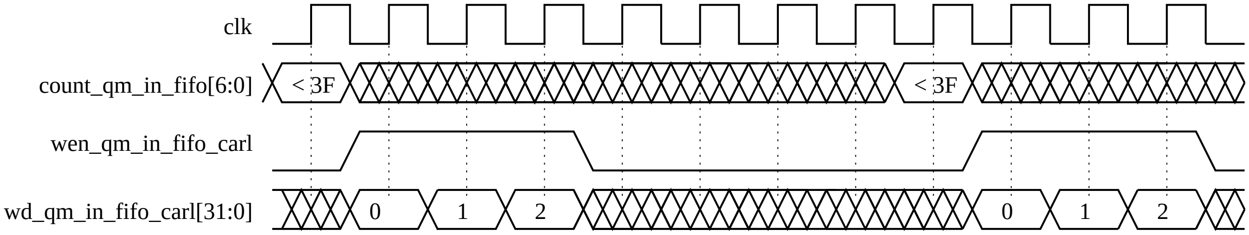


Figure 59: $carl_qm_timing$ Timing diagram for CARL/QMGR interface.

6.5 QMGR/OSAR Interface

Packet references are transmitted from the Queue Manager (QMGR) block to the Output Segmentation and Reassembly (OSAR) block following queueing and scheduling. The data frame format for the QMGR/OSAR interface is shown in Figure 60 with timing relationship shown in Figure 61. *soh_osar_qm* indicates the beginning of 4 consecutive cycles transfer of a packet reference in *data_osar_qm[31:0]*. Transfer may occur back-to-back.

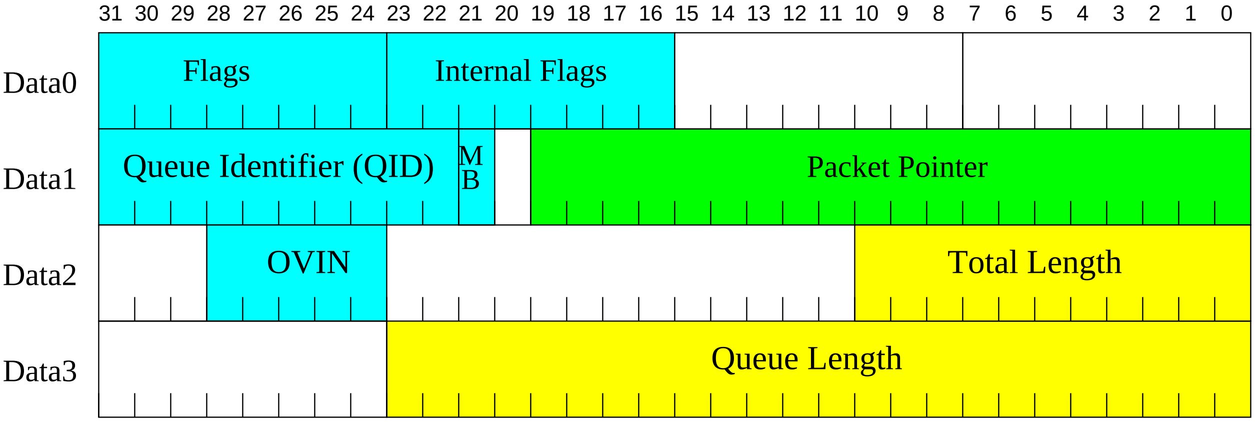


Figure 60: $qm_{o}sar_{f}rame$ Data frame format for QMGR/OSAR interface.

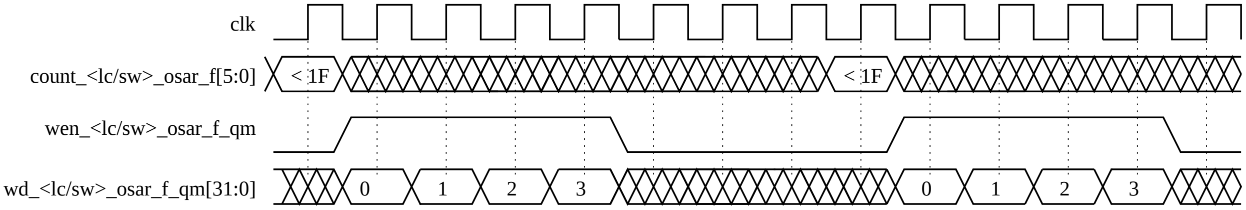


Figure 61: $qm_{o}sar_{t}iming$ Timing diagram for QMGR/OSAR interface.

6.6 QMGR/PSM Interface

The Queue Manager (QMGR) has a separate packet drop FIFO interface to the Line Card Packet Storage Manager (LC PSM) and the Switch Packet Storage Manager (SW PSM). The FIFO is 16 bits wide and 256 words deep. Each valid drop request consists of two 16-bit words. Each FIFO can store up to 128 drop requests. The Drop FIFO content for the QMGR/PSM interface is shown in Figure 62.

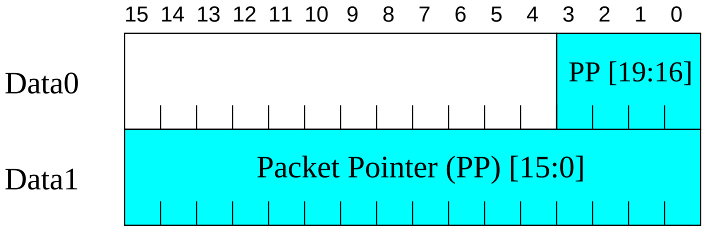


Figure 62: qm_psm_frame Drop FIFO content for QMGR/PSM interface.

- *packet pointer [19:0]* is the pointer to the first chunk of the packet.

6.7 OSAR/PSM Interface

The Onput Segmentation and Reassembly (OSAR) block interfaces to the onput side of the Packet Storage Manager (PSM). Packets that are ready to be sent to the LC or SW interfaces are requested by the OSAR, retrievd by the PSM and transferred to the OSAR in fixed size chunks. Each chunk carries up to 124 bytes of the packet. Figure 53 and Figure 54 shows the slight difference in format between chunks used as the first chunk and the n^{th} chunk. The first 3 bytes of a chunk is used by the PSM to store the next chunk pointer. The 4^{th} byte of the first chunk of a packet is used to carry the Input Virtual Identification Number (IVIN).

Chunks are transferred across a 64-bit data bus with associated control signals. The timing diagram for this interface is shown in Figure 63. These signals are explained below:

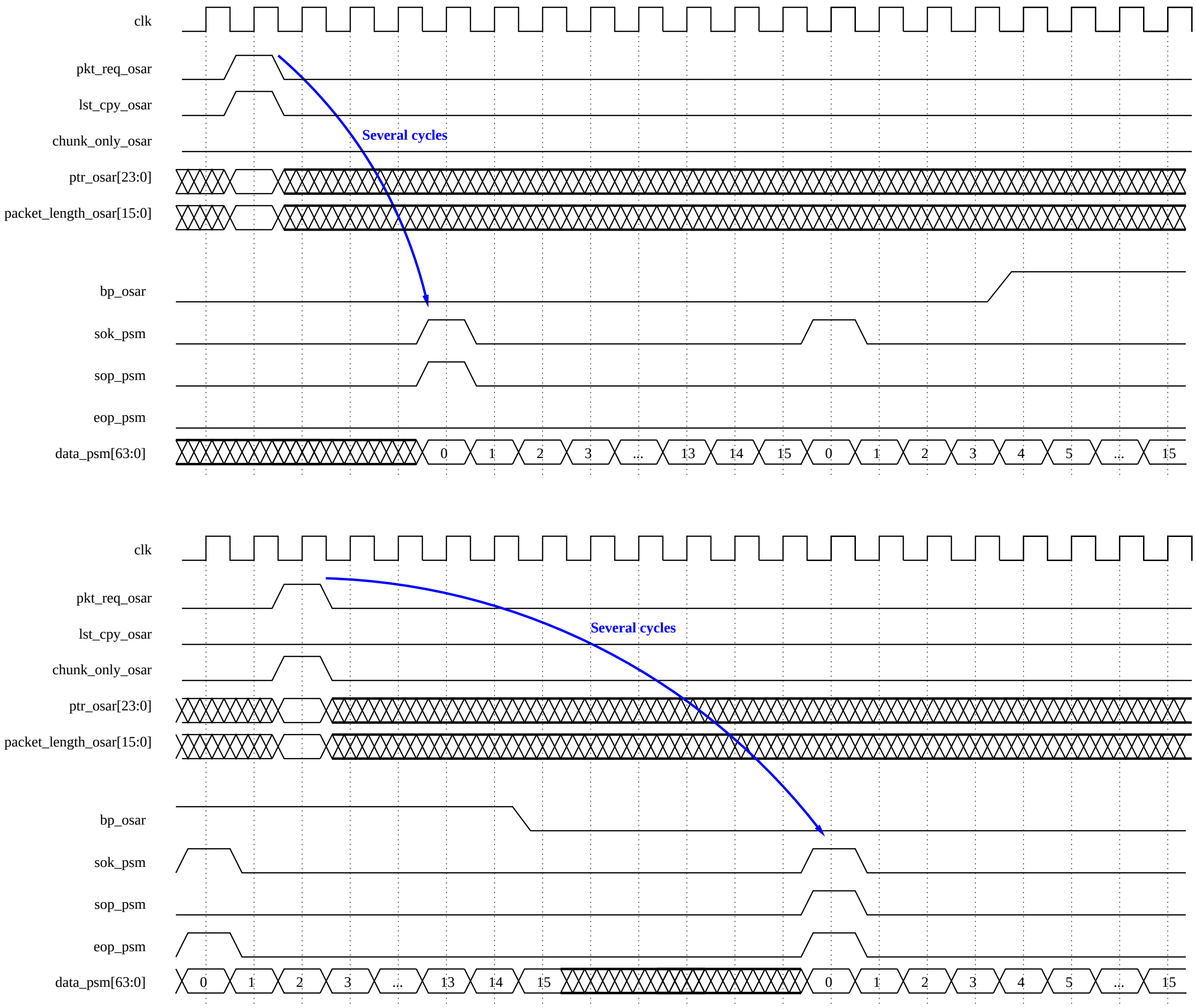


Figure 63: $osar_psm_timing$ Timing diagram for OSAR/PSM interface.

- *PKT_REQ_OSAR* indicates a request that a packet located at *PTR_OSAR* is to be retrieved. Specificity of the request is further defined by *LST_CPY_OSAR*, *CHUNK_ONLY_OSAR*, *PACKET_LENGTH_OSAR* signals.
- *LST_CPY_OSAR* indicates the retrieved the packet is the last copy of the packet. When the signal is

asserted high, the PSM should release all chunk pointers after retrieving the packet. This signal will be asserted 1 for all single copy packets and will be asserted 1 for the last copy of the multi copy packets. If the signal is 0, the PSM should not append the chunk pointers to the free list.

- *CHUNK_ONLY_OSAR* indicates that the PSM only needs to retrieve the first chunk of the packet. The PSM does not free the first chunk pointer in this case.
- *PTR_OSAR[23:0]* is the packet pointer to the first chunk of the packet in the SDRAM.
- *PACKET_LENGTH_OSAR[10:0]* is the total length in bytes of the IP packet to be read.
- *SOK_PSM* indicates the **s**tart **o**f a chun**k** sent to the OSAR.
- *SOP_PSM* indicates the **s**tart **o**f an IP **p**acket to the OSAR.
- *EOP_PSM* indicates the **e**nd **o**f an IP **p**acket sent to the OSAR.
- *DATA_PSM[63:0]* is the packet data sent to the OSAR.
- *BP_OSAR* allows OSAR to back pressure PSM. When asserted high, OSAR can receive no more than one more chunk. The chunk being presently retrieved by PSM can still be forwarded in its entirety, but no more.

6.8 ISAR/CCP Interface

Control cells are passed to the CCP from the ISAR via a FIFO input interface as shown in Figure 64.

- •*write_ccp_isar*: FIFO write signal, must be asserted high for each 32-bit word transfer of control cell data.
- •*data_ccp_isar[31:0]*: 32-bit data path for control cell.
- •*full_isar_ccp*: FIFO full signal asserted from CCP to ISAR.
- •*count_ccp_isar*: count of 32-bit words stored in 32x255 FIFO; for an entire cell to fit, the count must be less than 241.

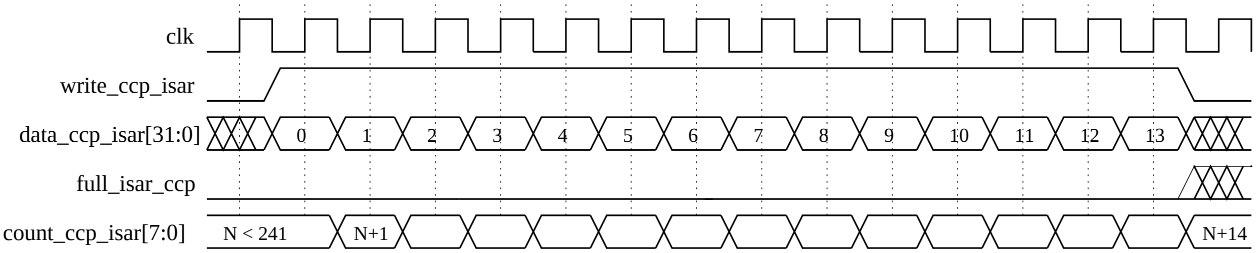


Figure 64: ccp_input_timing Timing diagram for the CCP cell input interface. This a FIFO style interface that writes to the CCP input FIFO.

ISAR increments the per-VCI input packet counters and packet drop counters (which are stored in the CCP) by passing the associated register number and a single clock cycle increment pulse as shown in Figure 65. Note that ISAR must wait a minimum of four clock cycles between successive increment commands. ISAR sets the Length Mismatch (LM) and IP Header checksum fail (IPH) at any time by pulsing the associate signal for a single clock cycle as shown in Figure 65.

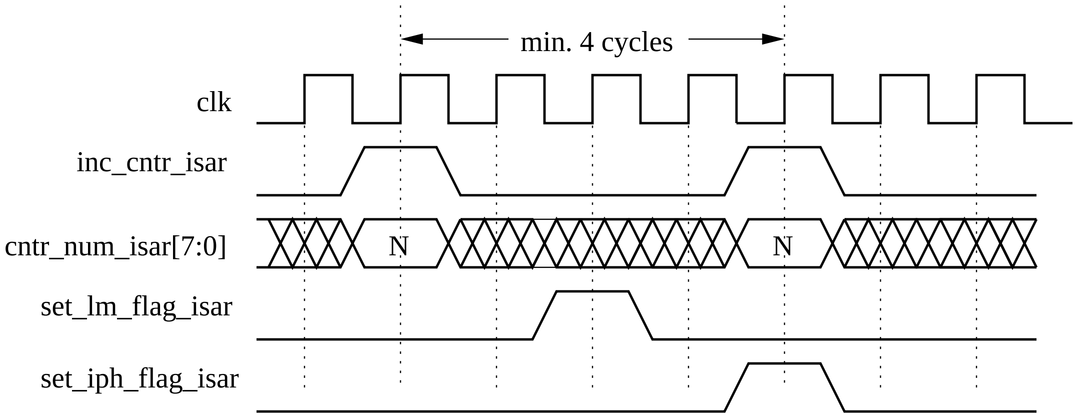


Figure 65: $ccp_isar_counter_timing$ Timing diagram for the ISAR counter increment interface to the CCP.

6.9 CCP/OSAR Interface

Control cells are transmitted to the OSAR via a modified UTOPIA interface as shown in Figure 66. In order to facilitate multiplexing control cells into the data path, the CCP first raises a send request signal, *send_req_osar_ccp*, to the OSAR when it has a control cell ready to transmit. OSAR services this request by raising the transmit cell available signal, *tca_ccp_osar*. Note that CCP has an 18 cell buffer, so control cells may be serviced by the OSAR at a low priority level. CCP signals the transfer of the control cell by asserting *soc_osar_ccp* on the first of fourteen consective 32-bit data word transfers, *data_osar_ccp[31:0]*. After the fourteenth word, CCP deasserts *send_req_osar_ccp*. Control cell formats vary, depending on the type of operation. CCP functions are described in Section 5.6.

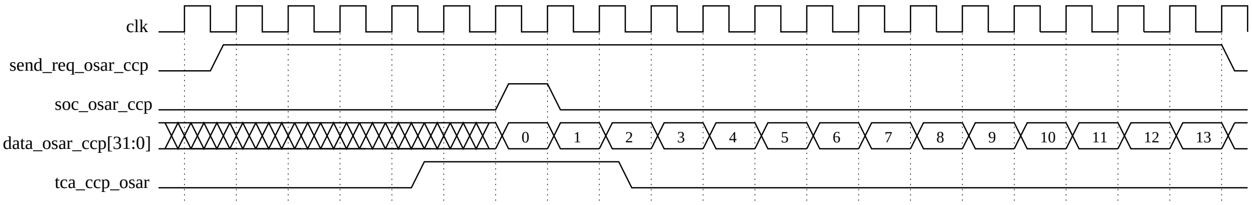


Figure 66: ccp_output_timing Timing diagram for the CCP cell output interface. This a UTOPIA style interface with the exception of the send request. When the CCP has a control cell response to send, it raises the send request line. The OSAR allows the cell to be transmitted by raising the transmit cell available signal.

OSAR increments the per-VCI output packet counters (which are stored in the CCP) by passing the associated register number and a single clock cycle increment pulse as shown in Figure 67. Note that OSAR must wait a minimum of four clock cycles between successive increment commands.

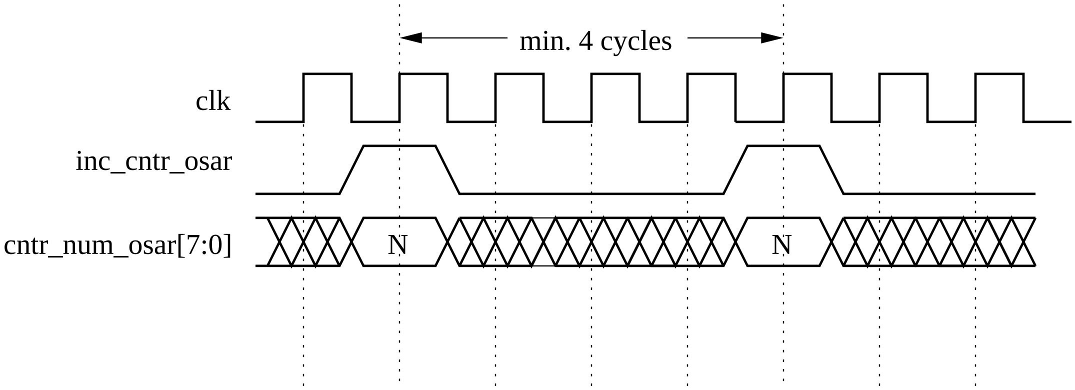


Figure 67: $ccp_osar_counter_timing$ Timing diagram for the OSAR counter increment interface to the CCP.

6.10 CCP/QMGR Interface

The following describes the interfaces between the Control Cell Processor (CCP) and the Queue Manager (QMGR). These interfaces allow control software to query the Queue Manager for queue lengths and exchange/update distributed queueing information.

6.10.1 QID Queue Length Query

The following describes the interface for a queue length query for a single queue identifier (QID). As shown in Figure 68, the CCP simply passes the 10-bit QID synchronous to a read signal. An unbounded time later, the Queue Manager responds with the 24-bit queue length synchronous to a valid signal.

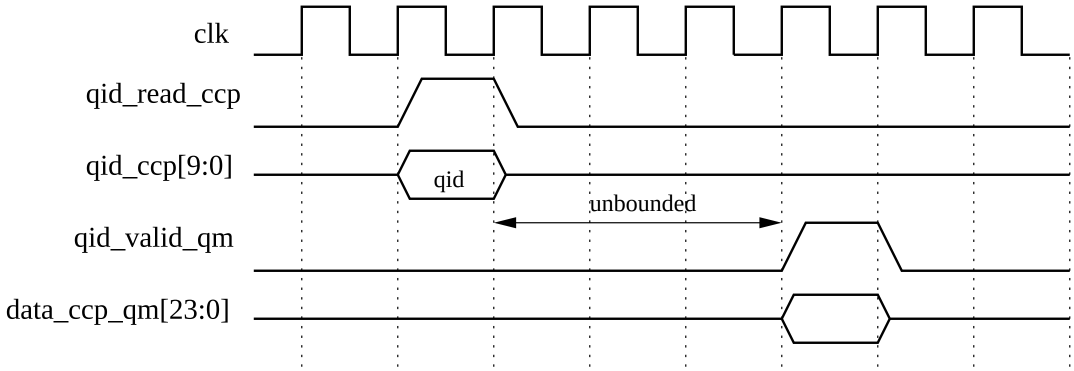


Figure 68: $ccp_q m_q id_t iming$ Timing diagram for QID queue length query interface between CCP and QMGR.

6.10.2 QID Quantum Updates

The following describes the interface for a queue length updates for a specific QID. As shown in Figure 69, the CCP asserts a request signal when the update control cell is received. After a variable number of clock cycles, the Queue Manager responds that it is ready to receive the update by asserting a grant pulse. After receiving the grant, the CCP writes one 32-bit word while holding the `qid_set_quantum_ccp` signal high.

6.10.3 QID Queue Length Threshold Update

The following describes the interface for a queue length threshold update for a specific QID. As shown in Figure 70, the CCP asserts a request signal when the update control cell is received. After a variable number of clock cycles, the Queue Manager responds that it is ready to receive the update by asserting a grant pulse. After receiving the grant, the CCP writes one 32-bit word while holding the `qid_set_qlen_ccp` signal high.

6.10.4 VOQ Queue Length Query

The following describes the queue length query for the set of eigth Virtual Output Queues (VOQ). As shown in Figure 68, the CCP simply issues a read request. An unbounded time later, the Queue Manager responds

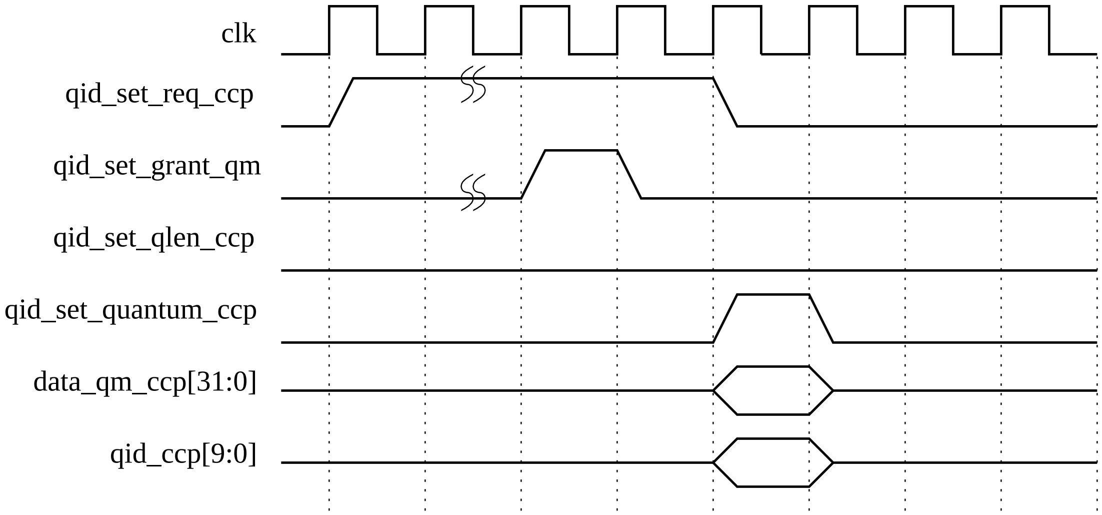


Figure 69: $ccp_qm_qid_quant_write$ Timing diagram for QID quantum update interface between CCP and QMGR.

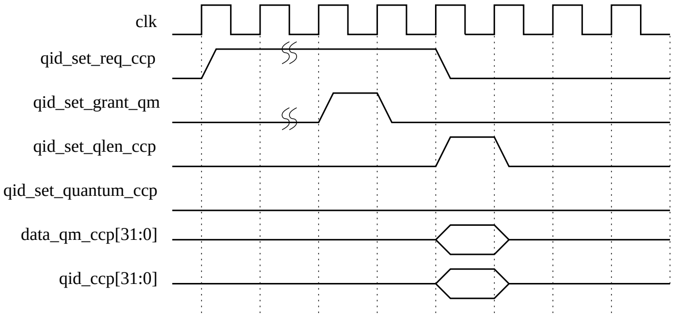


Figure 70: $ccp_qm_qid_qlen_write$ Timing diagram for QID queue length threshold update interface between CCP and QMGR.

with nine words containing 32-bit queue lengths (byte counts) for the eight VOQs and the 32-bit byte count for the sum of the egress queues synchronous to a valid signal.

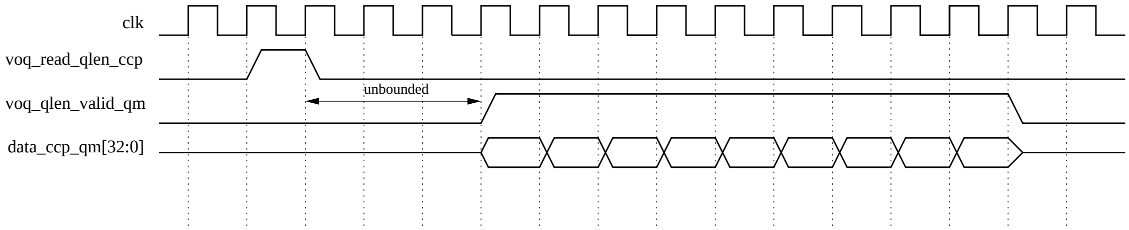


Figure 71: $ccp_qm_voq_timing$ Timing diagram for VOQ queue length query interface between CCP and QMGR.

6.10.5 VOQ Quantum Updates

The following describes the interface for updating the quantum (byte counts) used to schedule the eight virtual output queues (VOQs). As shown in Figure 72, the CCP asserts a request signal when the update control cell is received. After a variable number of clock cycles, the Queue Manager responds that it is ready to receive the updates by asserting a grant pulse. After receiving the grant, the CCP writes nine 32-bit words while holding the `voq_set_quantum_ccp` signal high. Note that the ninth 32-bit word contains the switch bandwidth as a rate expressed in Kbps.

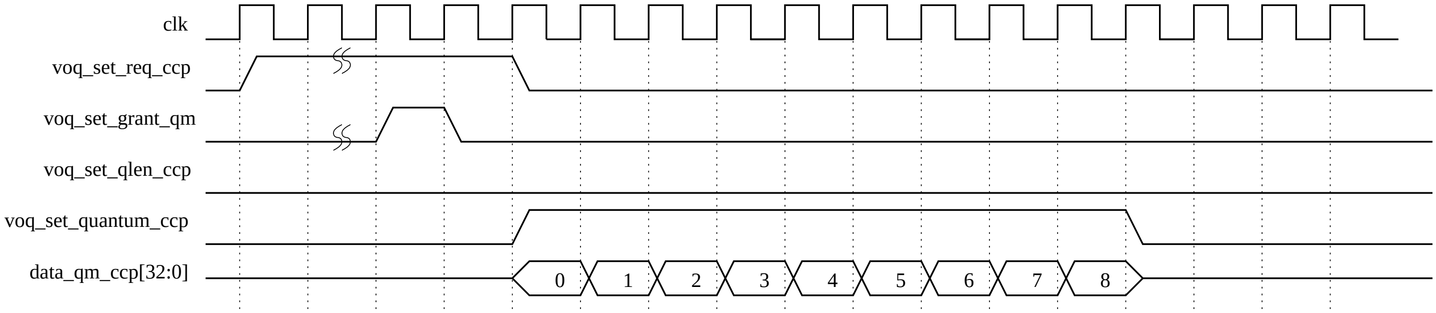


Figure 72: $ccp_qm_voq_quant_write$ Timing diagram for VOQ quantum update interface between CCP and QMGR.

6.10.6 VOQ Queue Length Updates

The following describes the interface for a queue length updates for virtual output queues (VOQs). As shown in Figure 73, the CCP asserts a request signal when the update control cell is received. After a variable number of clock cycles, the Queue Manager responds that it is ready to receive the updates by asserting a grant pulse. After receiving the grant, the CCP writes nine 32-bit words while holding the `voq_set_qlen_ccp` signal high.

6.10.7 Drop Counters

QMGR increments the packet drop counters (which are stored in the CCP) by passing the associated register number and a single clock cycle increment pulse as shown in Figure 74. Note that QMGR must wait a

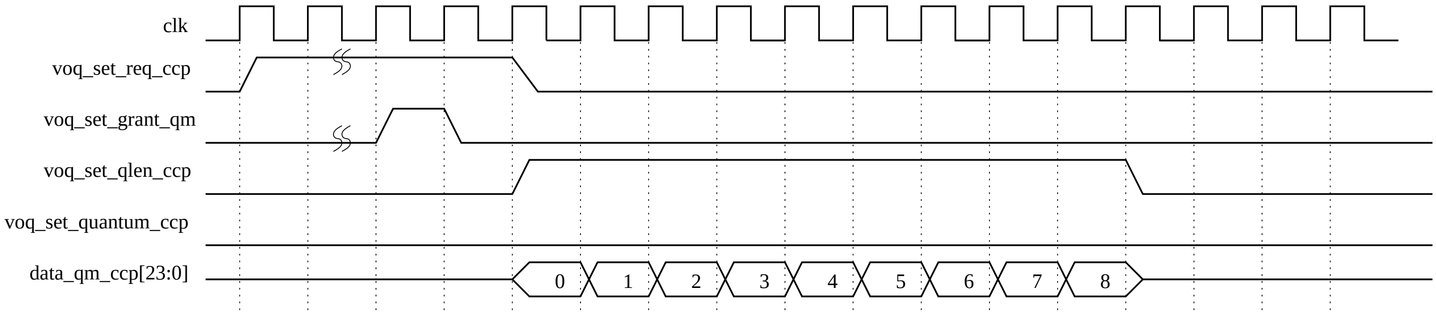


Figure 73: $ccp_qm_voq_qlen_write$ Timing diagram for QID queue length update interface between CCP and QMGR.

minimum of four clock cycles between successive increment commands.

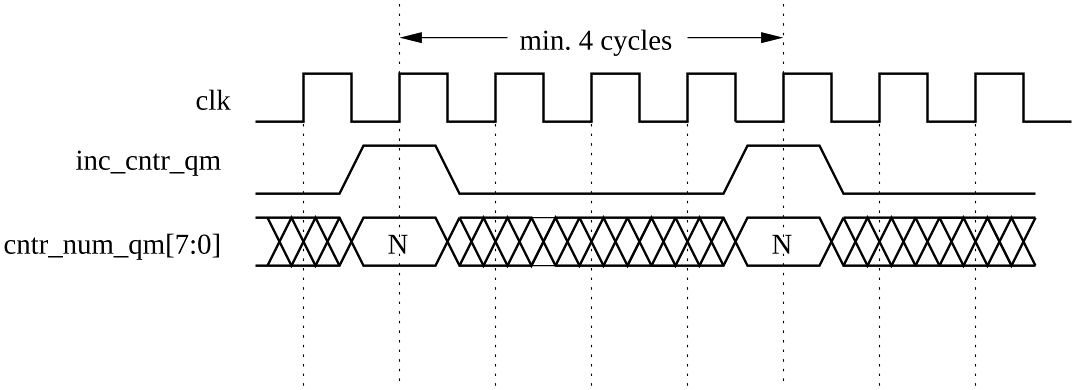


Figure 74: $ccp_qm_counter_timing$ Timing diagram for the QMGR counter increment interface to the CCP.

6.11 CCP/CARL Interfaces

Updates to the General Match filter table and Exact Match hash table are sent via control cells and processed by the Control Cell Processor (CCP). See Section 5.6 for update commands and control cell formats.

6.11.1 General Match Filter Table Updates

The General Match filter table is physically divided into two tables: the Key Table containing the filter key values used to match packet header fields and the Value Table containing the actions (QID, etc.) applied to packets matching the associated filter. Entries in both tables are addressed with a single Filter Identifier (FID). Each Key Table entry is comprised of two (2) 80-bit words (5x16-bits). Each Value Table entry is comprised of two (2) 32-bit words (2x16-bits). In order to conserve on-chip routing resources, the datapaths between the CCP and CARL are 16-bits wide.

The signals of the GM/CCP interface are defined below:

- **gm_filter_table_req**: GM filter table write request; CARL will stop processing new packets and wait until all packets in the pipeline and output FIFO have been processed before issuing a grant to ensure consistency between assigned FIDs and applied values
- **gm_filter_table_gr**: GM filter table write grant; CCP may begin the burst write of the Key and Value tables
- **gm_filter_table_fid(4:0)**: GM filter table Filter Identifier (FID); addresses 32 filter entries in Key and Value tables
- **gm_filter_table_data_wr(15:0)**: GM filter table write data
- **gm_filter_table_data_rd(15:0)**: GM filter table read data
- **gm_filter_table_data_we**: GM filter table write enable; held high (1) during 14 cycle write burst
- **gm_filter_table_data_re**: GM filter table read enable; held high (1) for 1 cycle with valid FID to issue a read command
- **gm_filter_table_data_rv**: GM filter table read valid; held high (1) for 10 cycles with valid Key Table data, then held high (1) for 4 cycles with valid Value Table data; there may be 0 to n cycles between data bursts where n is not bounded

The timing diagram for a GM Filter Table write transaction is shown in Figure 75. Note that after requesting write access, CARL responds with a grant. Prior to issuing the grant signal, CARL must stop feeding new packet headers to the GM engine and ensure that the GM processing pipeline and output FIFO are empty. This ensures correct binding of actions to assigned FIDs. Note that all data is transferred in a 14 cycle burst signaled by the write enable signal. The ordering of data is given below:

- **k0**: General Filter Key Word 1 [79:64]
- **k1**: General Filter Key Word 1 [63:48]
- **k2**: General Filter Key Word 1 [47:32]
- **k3**: General Filter Key Word 1 [31:16]
- **k4**: General Filter Key Word 1 [15:0]
- **k5**: General Filter Key Word 2 [79:64]
- **k6**: General Filter Key Word 2 [63:48]
- **k7**: General Filter Key Word 2 [47:32]
- **k8**: General Filter Key Word 2 [31:16]
- **k9**: General Filter Key Word 2 [15:0]

- **v0**: General Filter Value Word 1 [31:16]
- **v1**: General Filter Value Word 1 [15:0]
- **v2**: General Filter Value Word 2 [31:16]
- **v3**: General Filter Value Word 2 [15:0]

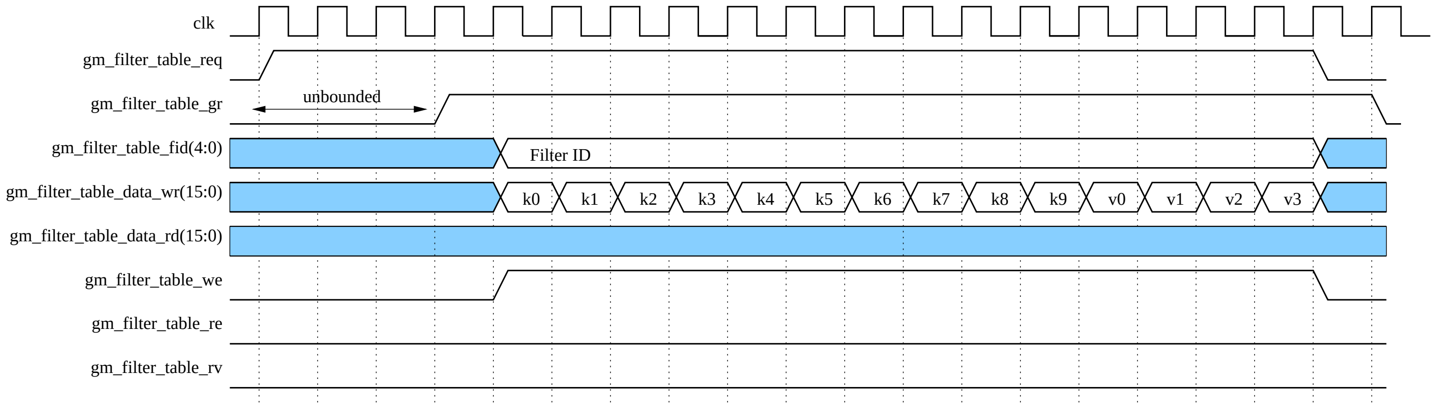


Figure 75: $ccp_carl_gm_write_timing$ Timing diagram for writing General Match filter entries.

The timing diagram for a GM Filter Table read transaction is shown in Figure 76. Note that no request/grant mechanism is used. CCP asserts write enable with the FID for one clock cycle. CARL may require a variable number of cycles to respond. CARL retains the data ordering listed above. Note that the ten 16-bit words of the Key Table entry are guaranteed to be a 10 cycle burst. The four 16-bit words of the Value Table entry may be transmitted a variable number of clock cycles later in a 4 cycle burst. Valid data is signaled via the read valid signal.

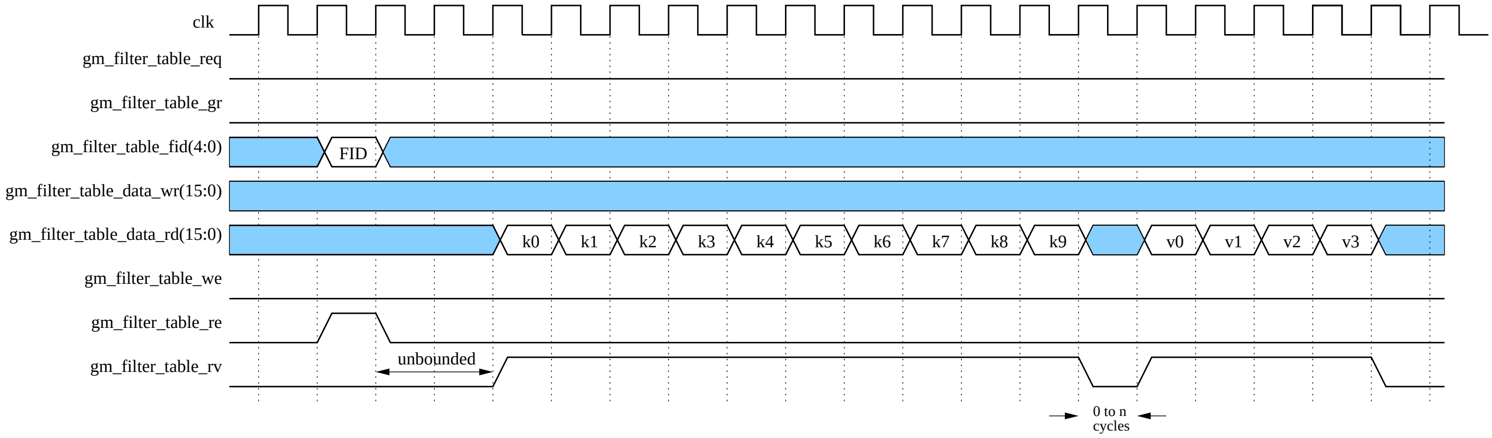


Figure 76: $ccp_carl_gm_read_timing$ Timing diagram for reading General Match filter entries.

6.11.2 Exact Match Hash Table Updates

The hash table used by the Exact Match search engine resides in a dual-ported on-chip BlockRAM configured with one read/write port and one read-only port. Since the CCP occupies the read/write port, no request/grant mechanism is required. Updates to the Exact Match datastructure must simply be issued in the correct order to ensure correctness. The timing diagram for EM Hash Table updates is shown in Figure 77.

Note that the read/write signal is asserted high (1) for a write. This signal must be asserted 2 clock cycles following a cycle when *em_hash_table_idle* is high (1).

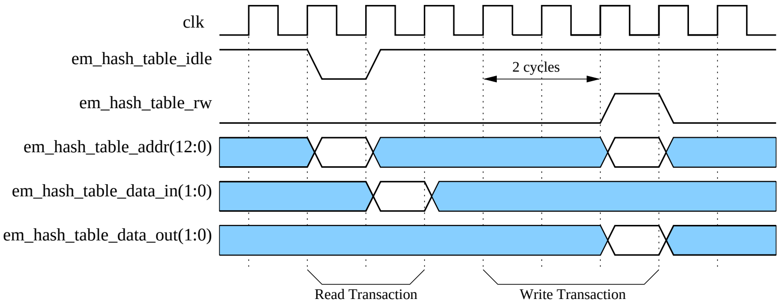


Figure 77: $ccp_carl_em_hash_table_timing$ Timing diagram for Exact Match hash table udpates; read transaction followed by a write transaction..

7 Block RAM Usage in the NSP RAD Circuit

The following tables show the details of where Block RAMs are used in the NSP RAD circuit. Table 13 gives the totals for the Block RAM use of the major modules in the NSP RAD circuit. A Block RAM in the VirtexE-E 2000 (xcv2000e-6), is 4Kbits arranged as 16b wide by 256 deep.

7.1 ISAR Block RAM Usage

Table 14 gives the details of the ISAR module's use of Block RAMs. There are two instances of the ISAR used in the RAD circuit, one for the Line Card (LC) side and one for the Switch (SW) side. Each uses its own instantiations of the same three components: CSB, PSB and Cell Fifo.

- CSB: Chunk Storage Buffer
  - Type: Dual Port RAM
    - Port A Width: 32b
    - Port A Depth: 2048
    - Port B Width: 64b
    - Port B Depth: 1024
  - Size: 16 Block RAMs
  - Notes on Size
    - need 4 Block RAMs wide to get 64b wide Port B
    - need 4 Block RAMs deep to get 1024 entry depth for Port B
  - Questions on Size
    - Does CSB have to be same size on Ingress (5 contexts) and Egress (9 contexts)?
    - Is the CSB allocated efficiently?
    - How is CSB Address formed?
      - variable vCSBAddr : std_logic_vector (10 downto 0)
      - CSBAddr : std_logic_vector (3 downto 0)
      - CSBSubAddr : std_logic_vector (6 downto 0)
      - vCSBAddr := CSBAddr & CSBSubAddr
      - CSBAddr sets the base address for a context
      - CSBSubAddr sets the current address to write to within a context.
      - Ingress: Seems like we will only use 5 base addresses out of 16 possible
      - Edgess : seems like we will only use 9 base addresses out of 16 possible
      - from file vci2cid.vhd, it seems that Ingress uses contexts: 4 (SPC), and 3-0 (Subports)
      - from file vci2cid.vhd, it seems that Egress uses contexts: 8 (SPC), and 7-0 (From Ports)

  So, it would seem that the CSB is grossly over allocated. We might be able to trim it down without having to specicialize the Ingress or Egress sides. Right now they both use the same addressing scheme and same length address signals. Keep in mind that there is one ISAR entity that is instantiated twice, once for Ingress and once for Egress. And there is currently one CSB defined, again with two instantiations, one for Ingress and one for Egress.

- PSB: Packet Storage Buffer

  - Type: Dual Port RAM
    - ∗ Port A Width: 32b
    - ∗ Port A Depth: 256
    - ∗ Port B Width: 64b
    - ∗ Port B Depth: 128
  - Size: 4 Block RAMs
  - Notes on Size
    - ∗ need 4 Block RAMs to get 64b wide Port B
    - ∗ seems like it could be twice as deep as we have it configured.

- Cell FIFO
  - Type: FIFO
    - ∗ Width: 32b
    - ∗ Depth: 256
  - Size: 2 Block RAMs
  - Notes on Size
    - ∗ need 2 Block RAMs to get 32b wide

7.2 CARL Block RAM Usage

Table 15 gives the details of the CARL module's use of Block RAMs.

- GM Filter Table (gm_ft1)
  - Type: Dual Port RAM
    - ∗ Port A: Read/Write
    - ∗ Port A Width: 80
    - ∗ Port A Depth: 256
    - ∗ Port B: Read/Write
    - ∗ Port B Width: 80
    - ∗ Port B Depth: 256
  - Size: 5 Block RAMs
  - Notes on Size
    - ∗ need 5 Block RAMs to get 80b wide Port A and Port B
- GM Results Table (gm_ft2)
  - Type: Dual Port RAM
    - ∗ Port A: Read/Write
    - ∗ Port A Width: 32b
    - ∗ Port A Depth: 256
    - ∗ Port B: Read/Write
    - ∗ Port B Width: 32b

    - Port B Depth: 256
  - Size: 2 Block RAMs
  - Notes on Size
    - need 2 Block RAMs to get 32b wide Port A and Port B

- GM in FIFO
  - Type: FIFO
    - Width: 64b
    - Depth: 256
  - Size: 4 Block Rams
  - Notes on Size
    - need 4 Block RAMs to get 64b wide

- GM out FIFO
  - Type: FIFO
    - Width: 16
    - Depth: 256
  - Size: 1 Block RAM
  - Notes on Size
    - need 1 Block RAMs to get 16b wide

- EM Hash Table
  - Type: Dual Port RAM
    - Port A: Read Only
    - Port A Width: 2b
    - Port A Depth: 8192
    - Port B: Read/Write
    - Port B Width: 2b
    - Port B Depth: 8192
  - Size: 4 Block RAMs
  - Notes on Size
    - the block rams are used as 1x4K primitives to get a 2x8K table.

- EM input FIFO
  - Type: FIFO
    - Width: 32b
    - Depth: 256
  - Size: 2 Block RAMs
  - Notes on Size
    - need 2 Block RAMs to get 32b wide

- EM output FIFO
  - Type: FIFO
    - Width: 32b
    - Depth: 256
  - Size: 2 Block RAMs
  - Notes on Size
    - need 2 Block RAMs to get 32b wide
- RL in FIFO
  - Type: FIFO
    - Width: 16b
    - Depth: 256
  - Size: 1 Block RAM
  - Notes on Size
    - need 1 Block RAMs to get 16b wide
- RL out FIFO
  - Type: FIFO
    - Width: 16b
    - Depth: 256
  - Size: 1 Block RAM
  - Notes on Size
    - need 1 Block RAMs to get 16b wide
- Bypass FIFO
  - Type: FIFO
    - Width: 32b
    - Depth: 512
  - Size: 4 Block RAMs
  - Notes on Size
    - need 2 Block RAMs wide to get 32b wide
    - need 2 Block RAMs long get 512 deep
- PktHdr FIFO
  - Type: FIFO
    - Width: 32b
    - Depth: 1024
  - Size: 8 Block RAMs
  - Notes on Size
    - need 2 Block RAMs wide to get 32b wide
    - need 4 Block RAMs long get 1024 deep

7.3 QMGR Block RAM Usage

Table 16 gives the details of the QMGR module's use of Block RAMs.

- qhdr table
  - Type: Dual Port RAM
    - Port A Width:
    - Port A Depth:
    - Port B Width:
    - Port B Depth:
  - Size: N Block RAMs
  - Notes on Size
    - need N Block RAMs to get xxb wide Port Y
- qlen table
  - Type: Dual Port RAM
    - Port A Width:
    - Port A Depth:
    - Port B Width:
    - Port B Depth:
  - Size: N Block RAMs
  - Notes on Size
    - need N Block RAMs to get xxb wide Port Y
- quantum table
  - Type: Dual Port RAM
    - Port A Width:
    - Port A Depth:
    - Port B Width:
    - Port B Depth:
  - Size: N Block RAMs
  - Notes on Size
    - need N Block RAMs to get xxb wide Port Y
- threshold table
  - Type: Dual Port RAM
    - Port A Width:
    - Port A Depth:
    - Port B Width:
    - Port B Depth:
  - Size: N Block RAMs
  - Notes on Size
    - need N Block RAMs to get xxb wide Port Y

7.4 OSAR Block RAM Usage

Table 17 gives the details of the OSAR module's use of Block RAMs.

- LC AAL5 dfifo
  - Type: FIFO
    - ∗ Width:
    - ∗ Depth:
  - Size:
  - Notes on Size
    - ∗ need N Block RAMs to get xxb wide
- LC AAL5 lfifo
  - Type: FIFO
    - ∗ Width:
    - ∗ Depth:
  - Size:
  - Notes on Size
    - ∗ need N Block RAMs to get xxb wide
- LC PB
  - Type: Dual Port RAM
    - ∗ Port A Width:
    - ∗ Port A Depth:
    - ∗ Port B Width:
    - ∗ Port B Depth:
  - Size: N Block RAMs
  - Notes on Size
    - ∗ need N Block RAMs to get xxb wide Port Y
- SW AAL5 dfifo
  - Type: FIFO
    - ∗ Width:
    - ∗ Depth:
  - Size:
  - Notes on Size
    - ∗ need N Block RAMs to get xxb wide
- SW AAL5 lfifo
  - Type: FIFO
    - ∗ Width:

        - Depth:
    - Size:
    - Notes on Size
        - need N Block RAMs to get xxb wide

- SW PB
    - Type: Dual Port RAM
        - Port A Width:
        - Port A Depth:
        - Port B Width:
        - Port B Depth:
    - Size: N Block RAMs
    - Notes on Size
        - need N Block RAMs to get xxb wide Port Y

7.5 CCP Block RAM Usage

Table 18 gives the details of the CCp module's use of Block RAMs.

- cell store
    - Type: Dual Port RAM
        - Port A Width:
        - Port A Depth:
        - Port B Width:
        - Port B Depth:
    - Size: N Block RAMs
    - Notes on Size
        - need N Block RAMs to get xxb wide Port Y
- counters table
    - Type: Dual Port RAM
        - Port A: Read Only
        - Port A Width: 16b
        - Port A Depth: 256
        - Port B: Write Only
        - Port B Width: 16b
        - Port B Depth: 256
    - Size: 2 x 1 Block RAMs
    - Notes on Size
        - implemented as two separate 16b wide block rams
        - need 2 Block RAMs to get 32b wide: upper_16 and lower_16
        - currently unclear to me (JDD) why it was implemented like this.

7.6 PSM Block RAM Usage

Table 19 gives the details of the PSM module's use of Block RAMs.

- LC Chunk Queue
  - Type: FIFO
    - Width:
    - Depth:
  - Size:
  - Notes on Size
    - need N Block RAMs to get xxb wide
- LC Request Queue
  - Type: FIFO
    - Width:
    - Depth:
  - Size:
  - Notes on Size
    - need N Block RAMs to get xxb wide
- LC Free Ptrs
- SW Chunk Queue
  - Type: FIFO
    - Width:
    - Depth:
  - Size:
  - Notes on Size
    - need N Block RAMs to get xxb wide
- SW Request Queue
  - Type: FIFO
    - Width:
    - Depth:
  - Size:
  - Notes on Size
    - need N Block RAMs to get xxb wide
- SW Free Ptrs

Table 13: Block RAM Usage Table

Module Name	*Number of Block RAMS*
CARL	34
CCP	4
ISAR	44(36)
OSAR	14
PSM	28
QMGR	20
OSAR in fifo(2)	4
PSM drop fifo(2)	2
QM in fifo	2
Total Used	152(144)
Total Available	160

Table 14: ISAR Block RAM Usage Table

Module Name	*Number of Block RAMS*
ISAR Cell Fifo LC	2
ISAR CSB LC	16(12)
ISAR PSB LC	4
ISAR Cell Fifo SW	2
ISAR CSB SW	16(12)
ISAR PSB SW	4
ISAR Total	44(36)

Table 15: CARL Block RAM Usage Table

Module Name	*Number of Block RAMS*
CARL GM filter table	5
CARL GM in fifo	4
CARL GM out fifo	1
CARL GM results table	2
CARL EM Hash Table	4
CARL EM in fifo	2
CARL EM out fifo	2
CARL RL in fifo	1
CARL RL out fifo	1
CARL Bypass Fifo	4
CARL PktHdr Fifo	8
CARL Total	34

Table 16: QMGR Block RAM Usage Table

Module Name	*Number of Block RAMS*
QMGR qhdr table	10
QMGR qlen table	3
QMGR quantum table	4
QMGR threshold table	3
QMGR Total	20

Table 17: OSAR Block RAM Usage Table

Module Name	*Number of Block RAMS*
OSAR LC AAL5 dfifo	2
OSAR LC AAL5 lfifo	1
OSAR LC PB	4
OSAR SW AAL5 dfifo	2
OSAR SW AAL5 lfifo	1
OSAR SW PB	4
OSAR Total	14

Table 18: CCP Block RAM Usage Table

Module Name	*Number of Block RAMS*
CCP cell store	2
CCP counters table	2
CCP Total	4

Table 19: PSM Block RAM Usage Table

Module Name	*Number of Block RAMS*
PSM LC Chunk Queue	8
PSM LC Request Queue	2
PSM LC Free Ptrs	4
PSM SW Chunk Queue	8
PSM SW Request Queue	2
PSM SW Free Ptrs	4
PSM Total	28

8 Areas for Improvements and Simplications of the NSP RAD Circuit

The following list notes where in the NSP RAD circuit improvements or simplications can be made. This list is intended solely for the designers to keep track of such things.

- Interport shim can be reduced from 8 to 4 Bytes. It can possibly go down to 2 bytes. This should impact only the ISAR and OSAR.
- Intraport shim can be reduced from 16 to 8 Bytes. This should impact only the ISAR, OSAR and the SPC kernel.
- We still have a register DQ Control VCI in the CCP and signals from the CCP to the ISAR and OSAR for supplying them with that value.
- There is now room in the exact match filter entry to make the packet counter and byte counter both 32 bits.
- Probably the MB bit can be removed. It probably only served a purpose when used in conjunction with the MTP field for multicast. This will impact QMGR, CARL, ISAR and OSAR.
- The CARL to QM frame can be reduced to 2 words with some re-arranging, if the MB bit can indeed be removed.
- The QM to OSAR frame can be reduced to 3 words with some re-arranging.
- The size of the Flags and Internal Flags that are passed between blocks can probably be made smaller which may the allow other frame sizes between blocks to be reduced.

References

[1] F. Kuhns, J. DeHart, R. Keller, J. Lockwood, P. Pappu, J. Parwatikar, E. Spitznagel, W. Richard, D. Taylor, J. Turner, and K. Wong, "Implementation of an open multi-service router," Tech. Rep. WUCS-01-20, Applied Research Laboratory, Department of Computer Science, Washington University in Saint Louis, August 2001.

[2] J. D. DeHart, W. D. Richard, E. W. Spitznagel, and D. E. Taylor, "The smart port card: An embedded Unix processor architecture for network management and active networking," Tech. Rep. WUCS-01-18, Applied Research Laboratory, Department of Computer Science, Washington University in Saint Louis, August 2001.

[3] J. W. Lockwood, J. S. Turner, and D. E. Taylor, "Field programmable port extender (FPX) for distributed routing and queuing," in *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2000)*, (Monterey, CA, USA), pp. 137–144, Feb. 2000.

[4] J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor, "Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX)," in *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2001)*, (Monterey, CA, USA), pp. 87–93, Feb. 2001.

[5] D. E. Taylor, J. W. Lockwood, and S. Dharmapurikar, "Generalized RAD module interface specification of the field-programmable port extender (fpx)," Tech. Rep. WUCS-TM-01-15, Applied Research Laboratory, Department of Computer Science, Washington University in Saint Louis, July 2001. Available on-line as `http://www.arl.wustl.edu/arl/projects/fpx/wugs.ps`.

[6] D. E. Taylor, J. W. Lockwood, and N. Naufel, "Rad module infrastructure of the field-programmable port extender (fpx)," Tech. Rep. WUCS-TM-01-16, Applied Research Laboratory, Department of Computer Science, Washington University in Saint Louis, July 2001.

[7] "Field Programmable Port Extender Homepage." Online `http://www.arl.wustl.edu/arl-/projects/fpx/`, Aug. 2000.

[8] J. S. Turner, T. Chaney, A. Fingerhut, and M. Flucke, "Design of a Gigabit ATM switch," in *INFOCOM'97*, 1997.

[9] N. Naufel and J. Lockwood, "FPX System Documentation Supplemental Details: Network Interface Device (NID) Specification," tech. rep., Washington University in Saint Louis, August 2000.

[10] W. N. Eatherton, "Hardware-Based Internet Protocol Prefix Lookups." thesis, Washington University in St. Louis, 1998.

[11] D. E. Taylor, J. W. Lockwood, T. S. Sproull, and J. S. Turner, "Scalable IP Lookup for Programmable Routers," Tech. Rep. WUCS-TM-01-33, Applied Research Laboratory, Department of Computer Science, Washington University in Saint Louis, October 2001.